A Flexible and Extensible Architecture for Experimental Model Validation

Stefan Leye
Institute of Computer Science
Albert-Einstein-Str. 21
18059 Rostock, Germany
stefan.leye@uni-rostock.de

Adelinde M. Uhrmacher
Institute of Computer Science
Albert-Einstein-Str. 21
18059 Rostock, Germany
adelinde.uhrmacher@uni-rostock.de

ABSTRACT

With the rising number and diversity of validation methods, the need for a tool supporting an easy exploitation of those methods emerges. We introduce FAMVal, a validation architecture that supports the seamless integration of different validation techniques. We structure a validation experiment into the tasks specification of requirements, configuration of the model, model execution, observation, analysis, and evaluation. This structuring improves the flexibility of the approach, by facilitating the combination of methods for different tasks. In addition to the overall architecture, basic components and their interactions are presented. The usage of FAMVal is illuminated by several validation experiments with a small chemical model. The architecture has been realized using the plug-in based design of the modeling and simulation framework JAMES II.

Categories and Subject Descriptors

I.6.7 [**Simulation Support Systems**]: Environments; I.6.4 [**Model Validation and Analysis**]; G.4 [**Mathematical Software**]: Algorithm design and analysis; G.4 [**Reusable Software**]

General Terms

Model validation, Simulation Experiment

Keywords

Validation, Analysis, Experiment, Flexibility, Software

1. INTRODUCTION

The importance of model validation is emphasized by a variety of publications spanning more than 2 decades [36, 2, 3, 8] of identifying problems [19, 23, 14] and suggesting approaches for solutions [4, 38, 9]. The result of these activities is a large number of diverse validation methods [37].

The plethora of validation methods alone calls for the design of a tool that seamlessly integrates these methods and makes them available to users. However, most modeling and simulation tools are constrained to single or specific methods, e.g., [17], and do not allow an easy extension and experimentation with different methods. In addition current studies revealed that a better support is required, as users often lack the required mathematical background for a systematic validation [29]. One way to tackle this problem is structuring the validation process and identifying specific steps that need recognition and support equally [27]. For instance, the argument that validation lacks the scientific rigorousness compared to verification [5] refers to the requirements specification, which thus should receive more attention. This is a crucial problem, since the role of validation in complementing verification methods is undisputed [3].

Therefore, we propose the flexible and extensible architecture FAMVal (Flexible Architecture for Model Validation) which has been realized using the modeling and simulation framework JAMES II. FAMVal exploits JAMES II's plug-in based structure [21, 20], and is supported by its extensive libraries of modeling formalisms, simulation algorithms, and analysis methods.

As a first step, we will specify the requirements for FAMVal in more detail. Afterward the validation process is structured and crucial tasks are identified. Furthermore, the design of FAMVal is described. The focus will be on how the different components and their interplay support the different tasks of the validation process and facilitate its documentation and reuse. Some central implementation design decisions are described. A small chemical model and different validation experiments in FAMVal shall serve as proof of concept. The paper will conclude with a short discussion of related work and an outlook.

2. SOFTWARE REQUIREMENTS FOR FAMVal

Perrone et. al. [29] state that "the level of complexity of rigorous simulation methodology requires more from networking researchers than they are capable of handling without additional support from software tools." This observation is likely to be applicable to any application field of modeling and simulation, and refers to any of the activities involved in modeling and simulation, including validation. The problem becomes obvious when inspecting the diversity of methods and strategies applicable for validation [37]. However, what requirements should a software tool fulfill to provide additional support for the user?

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8833
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8833

First, the architecture should facilitate the realization, integration, evaluation, and use of diverse methods so to offer potential users sound implementations of current state of the art validation techniques. One possibility to address this requirement is to exploit the plug' simulate concept of JAMES II which supports re-using parts of algorithms, extending, configuring, and evaluating algorithms [21].

Validation requirements often imply a certain class of methods to be used. For instance, in order to identify interesting points in the parameter space, parameter estimation methods are required [15], in order to test the sensitivity of the model for certain parameters a sensitivity analysis is necessary [35]. Making validation requirements explicit, enhances the scientific rigorousness of validation [5] and forms a basis for developing strategies that assist users in selecting suitable methods.

The concrete application of validation methods requires typically an in-detail knowledge about methods and parameters to be specified. For instance, a fractional-factorial design for a sensitivity analysis, comprises parameters to denote fraction, resolution, parameters for aggregating the simulation output (e.g., by retrieving the steady-state), count of replications, required simulation end times, etc. However, often modelers might not be able or interested in specifying each single parameter of each single method used during the experiment, and prefer a more abstract view. For instance, one modeler might want to compare his reimplementation of a model in a new formalism with the original by comparing detailed simulation trajectories, while another modeler might want to compare his model with wet-lab data providing more abstract information (e.g., the occurrence of cycles). Hence, the validation architecture should support different abstraction levels for the specification and suitable means to translate more abstract experiment views into a concrete format. In [17] for instance, a domain specific language facilitates the specification and analysis of simulation experiments; comparatively abstract specifications created by the user are translated into a concrete format used for analysis.

The latter plays a central role in the envisioned tool, as the format will hold the information for executing a wide range of validation experiments. Thereby, it should contain the most detailed description of the experiment, in order to properly configure and control the single steps. It also forms the core of the experiment documentation and helps ensuring the repeatability of validation experiments. As new validation methods are being developed, extensions of the format might prove necessary. Thus, it should be possible for the format to be adapted to new needs and to accompany the entire validation process with its different tasks.

3. THE SIX TASKS OF A VALIDATION EXPERIMENT

In order to structure the architecture, typical tasks of a validation experiment have to be identified. This structuring allows to combine different methods facing individual tasks in a flexible manner. We distinguish six tasks in a validation experiment [27].

The first task is the specification of requirements for the model to be validated. Depending on the goal of the validation the requirements can lead to experiments ranging from simple (like comparing a single simulation trajectory with given data) to complex (like sweeping the parameter space of a model). Hence, the format to represent the requirements should be adaptable to various experiment designs. In addition, a formal description is essential in order to make it repeatable and comparable. Since, the requirements comprise crucial information (like applicable methods) that are of relevance for the following tasks, this step is the foundation of the validation experiment.

The second task is the configuration, which corresponds to the generation of test cases in software testing. Here points in the model parameter space are selected, that need to be investigated in order to achieve the desired information about the validity of the model. Those parameters depend on the model, e.g. they could be rates of chemical equations as well as a family of model components that represent the cell nucleus. The type of the parameter space also has an impact on the optimization methods used.

The third task is the execution of the model. Since accuracy (e.g. approximation issues of numerical ODE solvers) as well as bugs in the implementation can bias the simulation results, it is important to investigate different simulators and their impact. To identify interferences between specific model and simulator parameters, a separated configuration of both is required.

The fourth task is the observation, where the simulation output results are retrieved. The challenge during this step is to collect as much information as necessary to allow a proper analysis of the results but as little information as possible to save memory as well as computation costs during the analysis. It is for instance not necessary to observe all the variables of a model, if only one variable is of interest.

The analysis of the observed data is the fifth task, sometimes called runtime verification [6]. This part comprises two steps: the analysis of a single simulation run (e.g., calculation of the steady state) and the analysis of a set of replications (e.g., mean of the steady states of different replications). In general, the second step is based on the results of the first one, but occasions exist where just one of them is necessary. If for instance no stochasticity exists in the model and one replication is sufficient, no additional analysis of the replication is required. On the other hand, if trajectories of different replications have to be compared directly to calculate the monte-carlo variability [28], the single trajectories do not have to be analyzed. Various methods exist for both steps [16, 34, 7, 12]. However, they have to be chosen carefully in order to produce the proper information for the final task.

During this sixth task, i.e., the evaluation, the analysis results are used twofold. On the one hand feedback is produced for the configuration in order to identify additional interesting parameter combinations. On the other hand the result of the validation experiment are presented, this may be for instance the sensitivity of the parameters of the model or the preparation of the data as a figure to allow a face validation.

4. AN ARCHITECTURE FOR EXPERIMENTAL MODEL VALIDATION

As described in the previous sections, a validation tool should support solutions for the six validation tasks. Those solutions have to be arranged in a flexible, extensible, and configurable way, to offer the user the ability to apply the

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8833
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8833

right method at the right time as well as to perform repeatable and comparable experiments. We propose the validation architecture FAMVal based on the plug-in structure of James II [21, 20], which features flexibility, extensibility, and configurability. In the following we will present this architecture and show how its components interact in order to tackle the six tasks of a validation experiment.

4.1 Validation Requirements

The specification of requirements is a crucial point of each validation experiment. To represent requirements as well as the necessary information to execute validation experiments, we use *ParameterBlocks* in JAMES II, which can be utilized to configure models and algorithms alike. Each ParameterBlock is labeled with an identifier, holds a reference to an object representing a value associated to the identifier, and contains a set of sub-blocks, which are ParameterBlocks themselves. This design allows a simple creation of parameter configurations for hierarchical structures, by associating each level of the structure with the corresponding level of the ParameterBlock. So far, this way of representing parameters has been used to configure components realized as plug-ins in JAMES II. For instance, the ParameterBlock for a simulator using an event-queue, comprises a label denoting the simulation algorithm, as well as a sub-block responsible for the configuration of the event-queue.

As explained in Section 2 the base for specifying requirements is a structure that can store the most detailed description of the experiment configuration. ParameterBlocks are thus well suited to represent specified requirements, due to their high flexibility and adaptability. This can be illustrated by a ParameterBlock, used to configure a validation method comparing a simulation output trajectory and a given trajectory. The topmost block holds the reference trajectory as well as a label referring to the comparison method. In many cases this methods relies on an interpolation algorithm to prepare the simulation trajectory in order to synchronize its time points with those of the reference trajectory. Therefore, a sub-block is required, referring to the interpolation algorithm. Typically, this algorithm has its own parameters (e.g., size of splines), which leads to sub-blocks of the sub-block.

At the end of the specification phase, the ParameterBlock represents a detailed hierarchical description of the experiment setting, which is used to control the validation experiment. The repeatability of the validation experiments is assured, since modelers can exchange ParameterBlocks, reuse them to repeat the experiments and compare their results. Furthermore, the documentation of a validation experiment is facilitated, since the configuration of the methods is reflected in the ParameterBlock. To be more easily accessible by the user, a higher level validation specification language should be designed which is then translated into ParameterBlocks for execution.

4.2 Coordination of the Validation Experiment

According to the created ParameterBlocks representing the requirements, the class *BaseValidator* coordinates the tasks configuration, simulation execution, observation, analysis, and evaluation (see Figure 1). To structure the coordination and to facilitate a parallel execution of the experiment, the execution of those tasks is delegated to different components. Configuration and evaluation are tightly coupled, due to their interplay of creating configurations and giving feedback about the behaviour of those configurations. Therefore, the interaction of both tasks is handled by one component, the *ConfigurationSetValidator*. Each requirement represented by a ParameterBlock contains the information to configure a ConfigurationSetValidator and its associated components. Based on those information, the BaseValidator creates a ConfigurationSetValidator which receives the requirement, the location of the model, as well as a list of available simulation engine configurations. This list is used to parameterize the simulator using different simulator settings. As a result, bias produced by simulator components can be identified and therefore reduced (see Section 4.3). The ConfigurationSetValidator creates a set of *VariablesAssignments*, representing simulation configurations (including model and simulator parameters) that need to be executed. The execution and observation of those configurations, is realized by an external simulation tool, which uses the VariablesAssignments for executing the simulation runs.

The BaseAnalyzer is the central component for the analysis of the simulation output data and has to communicate to the simulation tool in order to inform it about required simulation end times, replications, and to retrieve the observed data in the right format for the analysis. Furthermore, the analysis results from the BaseAnalyzer are passed to the corresponding ConfigurationSetValidator, to activate the evaluation. For a detailed overview over the interaction of the components see Figure 2.

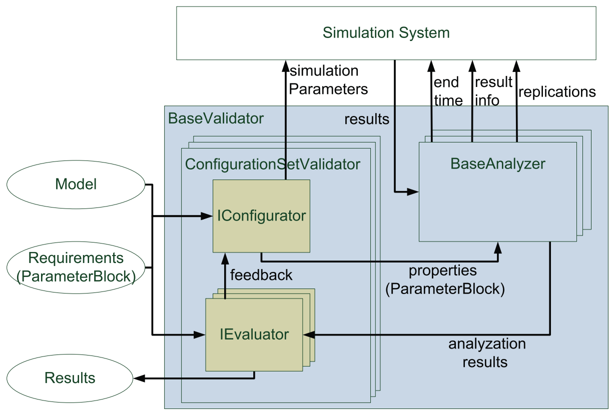

Figure 1: Scheme of a BaseValidator.

4.3 Configuration and Evaluation

The ConfigurationSetValidator is responsible for the configuration and evaluation task of the validation experiment. Therefore, it comprises two types of components, *Configurator* and *Evaluator*. Both components are coupled to the ConfigurationSetValidator by interfaces to maximize the flexibility of the possible functionality. While those interfaces constrain the structure of input and output of the methods in order to ensure a proper communication with the other components, the concrete functionality of the implemented method is not constrained. Furthermore, the components are realized as JAMES II plug-in types, which offers the opportunity to extend as well as to reuse the existing

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8833
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8833

functionality at this point. In order to follow the specification of the validation experiment, Configurator and Evaluator are configured according to the ParameterBlock given by the BaseValidator and representing the corresponding requirement.

To create the required VariablesAssignments for the BaseValidator, the ConfigurationSetValidator retrieves the model parameter configurations from the Configurator and merges them with the list of simulator parameter configurations. Thereby, each combination of available model and simulator configuration is created. This allows the Evaluator to investigate the influences of the different simulator configurations on the behaviour of each model configuration. In addition to the VariablesAssignment, the Configurator creates a set of ParameterBlocks (usually retrieved from the requirement ParameterBlock) representing the desired properties for the specific configuration. Those properties are required to configure the analysis step (see Section 4.4). So far, we implemented a simple Configurator, processing a list of given VariablesAssignments as well as more complicated Configurators for fractional-factorial or bifurcation experiment designs [25]. At the moment we are working on a Kriging [26] component and different Configurators to realize parameter estimation strategies.

After the simulation runs of a configuration have been executed and the output data has been analyzed, the evaluation process is started. The results of the analysis as well as the corresponding parameter configurations are passed to the Evaluator, which returns two results. Firstly, it produces the feedback for the Configurator in order to identify new interesting model parameter configurations. Secondly, it returns a list of simulator configurations, that turned out to be invalid (by biasing the results of the simulation runs) during the evaluation process. These configurations are removed from the BaseValidator's list of simulator configurations to prevent a further use of them. After all configurations have been executed and analyzed, the Evaluator conducts the final steps of the evaluation process, e.g., creating a figure for a face validation or writing evaluation results to a database. It is possible to create multiple Evaluators for one Configurator, to allow different evaluation strategies for the same set of configurations, whereas not every Evaluator implementation is suitable to interact with any Configurator. It was necessary for instance to create a corresponding Evaluator for each of the Configurator implementations mentioned before to realize the accordant design. On the other hand we created an Evaluator wrapping a Mosan [39] visualization, which does not depend on a Configurator and can interact with all of them.

The division between Configurator and Evaluator allows the flexible integration of diverse algorithms that chose interesting parameter configurations. E.g., in addition to the methods implemented so far, an optimization algorithm may be integrated easily by creating a Configurator that calculates the configurations and returns them as VariablesAssignments and an Evaluator, that gives informs the Configurator about successful configurations based on the simulation analysis results.

4.4 Analysis

The BaseAnalyzer is responsible for the coordination of the analysis process. As described in Section 3, this task is divided into two steps, the analysis of a single simulation run and the analysis of a set of replications. Both steps comprise loops of analysis and execution, to enable a dynamic adaptation of required simulation end times (for the first step) and replications (for the second step) in order to create sufficient simulation results for a proper analysis. To reflect the distinction of the steps, the BaseAnalyzer comprises the two components, *RunAnalyzer* executing the first, and *ConfigurationAnalyzer* executing the second one. Similar to the Configurator and Evaluator, both components are coupled to the BaseAnalyzer by interfaces and have been realized as plug-in types, to enable the most possible flexibility, extensibility and reusability of the used methods. The analysis highly depends on the properties, that have to be checked. Those properties have been created by the Configurator (see Section 4.3). Since usually, they are an integral part of the requirements and contain information about the experiment configuration, they are represented as ParameterBlocks. They are used to configure the RunAnalyzer and ConfigurationAnalyzer components.

Since, the output of a simulation run may be analyzed for a set of properties (e.g., steady-states of different variables), multiple RunAnalyzers may be required during an experiment. To structure their interaction and to facilitate a parallel execution, all the RunAnalyzers analyzing the same single simulation run, are coordinated by a *ReplicationAnalyzer*. When the simulation run has reached its tentative end time and output data are available, this component is responsible for the initiation of the analysis process of its RunAnalyzers. If a RunAnalyzer is not able to produce a clear result, it returns the estimated end time of the simulation run required to process a reliable analysis. The maximum of these end times, over all RunAnalyzers is the next tentative end time of the simulation run, which is forwarded to the simulation tool. Thereby, a dynamic and automatic adaptation of the simulation end time is possible, corresponding to the specific needs of the analysis step. This loop is repeated until none of the analyzers estimates a higher end time than the actual tentative end time of the simulation run. So far, we created eight different RunAnalyzers, including an equilibrium analyzer as well as different analyzers to compare simulation trajectories with given ones. Currently, we are working on an analyzer checking LTL-formulae in trajectories, to support simulation-based model-checking [16].

If a simulation run and its analysis have been finished, the results are passed to the corresponding ConfigurationAnalyzers to trigger the second step of the analysis. The results are processed with respect to the property and the amount of additionally required replications (leading to additional ReplicationAnalyzers as well as associated RunAnalyzers) is determined. If no additional replications are required the analysis results of the ConfigurationAnalyzer are created. The ConfigurationSetValidator uses them to trigger the evaluation. So far, we implemented an analyzer, calculating statistical measures (e.g., mean, variance,..) based on the results from the RunAnalyzer. We are working on an analyzer to calculate the monte-carlo variability of a model as well as a component corresponding to the RunAnalyzer checking for LTL-formulae in order to realize probabilistic simulation-based model-checking [12]. Thereby, the domain of the variables for a given LTL-formula and simulation run is created by the RunAnalyzer and forwarded to the ConfigurationAnalyzer, which calculates the accordant probabilistic domain for a set of simulations. While this analysis method

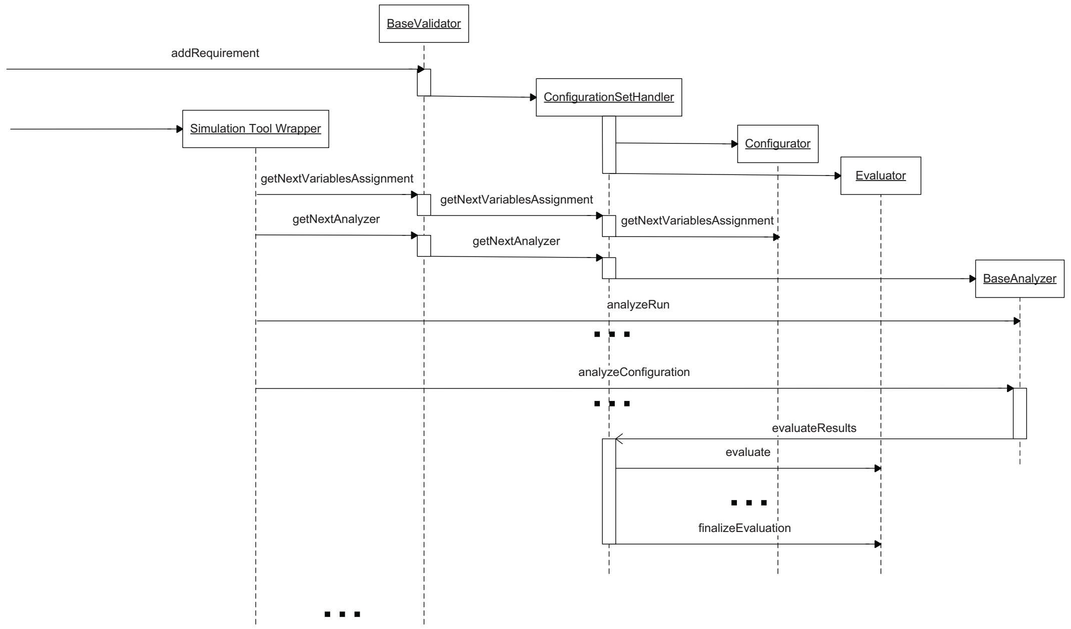

Figure 2: Sequence chart for the interaction of the BaseValidator with the associated components.

clearly differs from the ones implemented so far (which basically aggregate a trajectory into one value and calculate the mean over a set of simulations), it is well supported by the separation between the two analysis steps and their interaction.

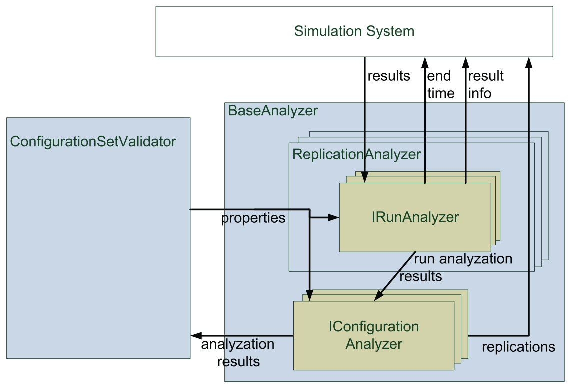

Figure 3: Scheme of a BaseAnalyzer.

Besides the required end time and the count of replications, the external simulation tool needs information about the type and format of the data to be observed. RunAnalyzers put constraints on the data they can analyze. These constraints are represented by an interface providing information about the variable name that is in the focus of the analysis, as well as the according data format (e.g., a list of double values for a trajectory). Objects implementing this interface are created for each RunAnalyzer. They are forwarded to the simulation tool using them as base for the instrumentation and therefore observation of the simulation runs.

4.5 Connection to a Simulation Tool

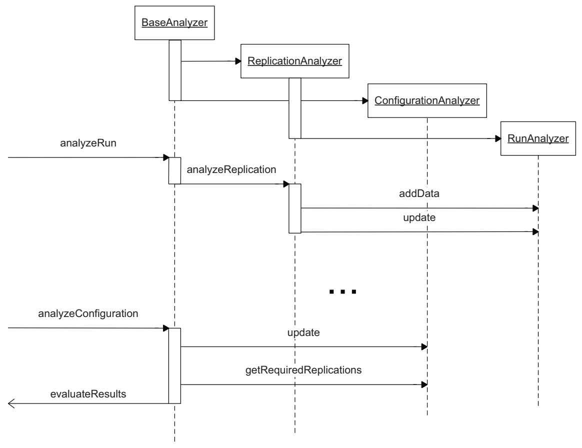

Figure 4: Sequence chart for the interaction of the BaseAnalyzer with the analysis components.

A vital part of a validation experiment is the execution of the model. FAMVal is based on the plug-in based structure of JAMES II and so far JAMES II is the only simulation tool supporting *all* the required features (including a strict distinction between model and simulation engine, flexible configuration of both, adaptable simulation run lengths and replications, flexible instrumentation of the simulation runs, etc.). However, we decided to offer the opportunity of exchanging the simulation tool. Thereby, the validation architecture offers the functionality to configure and to analyze the simulation runs, while a wrapper is necessary to manage the communication between validation and simulation tool. This strategy offers several advantages. The user can decide which simulation tool he uses, depending on his preferences and what is at hand. Furthermore, more than one simulation tool can be used, which is beneficial if one tool does not offer all the required functionality (e.g., support of different simulator components) and to compare results between tools. Finally, the simulation tool does not have to be a simulation tool literally. The source for simulation results could be for instance, a database holding the simulation output data. This could save computation costs, since it is not necessary to repeat simulation runs if their results are available already.

A wrapper connecting a simulation tool to FAMVal has to fulfill several tasks. It needs to retrieve the VariablesAssignments representing the simulation configurations from the BaseValidator, and has to forward them to the simulation tool, in a format the system can understand. Furthermore, it needs to pass the information about the required data format of the simulation output to the simulation tool to enable a proper observation. Correspondingly, the simulation output results have to be retrieved in order to execute the analysis process. Thereby, information about required simulation end times and replications have to be communicated to the simulation tool. In order to interact with FAMVal the simulation tool has to support a dynamic adaptation of simulation end times and replications, as well as a flexible system of instrumenting simulation runs. Since the interaction between FAMVal and a simulation tool is coordinated by a wrapper, the fact which component is the active part in the communication is kept flexible. A mix of the active and passive part is possible, e.g., the wrapper could actively push new simulation configurations to the simulation tool, while the tool on the other hand actively pushes the simulation output results to the wrapper, to trigger the analysis. Thus, the communication scheme between FAMVal and the simulation tool can be adapted flexibly to the needs of the tool. So far, we implemented a wrapper, based on *ExperimentVariables* to communicate with the *BaseExperiment* [20] of JAMES II. However, wrappers for additional simulation tools are possible, e.g., for the statistical framework of the ns-3 network simulator [1]. The wrapper would have the role of the experiment controller translating the VariablesAssignment created by FAMVal in order to spawn simulation instances. Furthermore it would configure the data collection according to the SimulationResutInfo objects given by FAMVal (and the used analysis methods), while each additional replication required for the data analysis leads to a new simulation instance. The only missing feature of the the statistical framework is the dynamic determination of simulation end times. However, this could be compensated by creating additional longer runs.

5. VALIDATION EXPERIMENTS

As mentioned in the previous section, we already created plug-ins for the components providing the validation methods. In the following we want to show, how the validation system is used, how these components interact, and how ParameterBlocks need to be configured, in order to realize different validation experiments. Thereby, the focus of this section does not lay on the experiments themselves, but on the benefits of the validation architecture used to configure and execute them.

5.1 Example model

During all of the experiments we use a model which represents reactions of a $MgCl_2$ solution. Originally, the model was introduced as a sample model for the stochastic π-calculus simulator SPiM [31]. As among other formalisms, JAMES II also supports the stochastic π-Calculus, it was easy to adapt for our purpose. The model comprises five species definitions Mg, Mg^+, Mg^{2+}, Cl, and Cl^- and four reactions with names $i1$, $i2$, $d1$ and $d2$ for ionization and de-ionization of Magnesium, respectively. The rates of the reaction are the parameters of the model.

At the beginning of each simulation run the initial amount of Mg and Cl particles is set to 100. During the experiments, the count of the different particles over time is observed.

5.2 Simple Validation

The first experiment has been executed using the *SimpleConfigurator*, processing a list of given model configurations. The parameters for this configurator contain a map of VariablesAssignments and associated ParameterBlocks representing the properties (see Figure 5). We executed the experiment, to measure the distances between the simulation results of the JAMES II example model and the source model executed with SPiM [32]. Therefore, we used the given parameter setting proposed in the description of the model (i1=10.0, i2=100.0, d1=50.0, d2=5.0), to create one VariablesAssignment. To represent the properties, we created a ParameterBlock (see Figure 6), including a string referring to the desired analysis method, the name of the variable that should be analyzed, as well as a reference trajectory (for the comparison), retrieved from a SPiM simulation run. We chose four methods to compare simulation trajectories, calculating the minimum, maximum, average, and average absolute deviation (all of the methods have been realized as RunAnalyzers). For the ConfigurationAnalyzer we added a label referring to the *StatisticalAnalyzer*, to calculate mean and variance of the deviations.

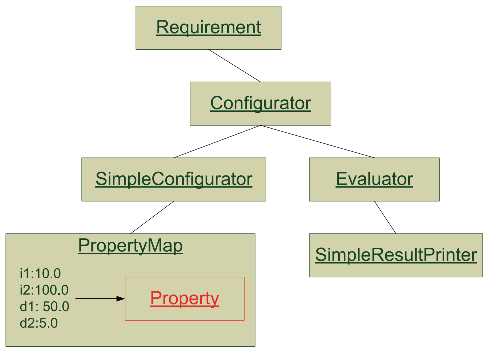

Figure 5: ParameterBlock for the simple validation experiment.

For a proper comparison of the models it is necessary, to include some additional considerations like testing different parameter settings. However, the focus on this experiment is on the proof of concept of the validation architecture, which is why we refrain from a broader investigation here.

The results of the experiment have been created by the *SimpleResultPrinter*, an Evaluator implementation, writing the received analysis results to a CSV-file. Table 1 shows, the content of that file. This variant leaves the configura-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8833
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8833

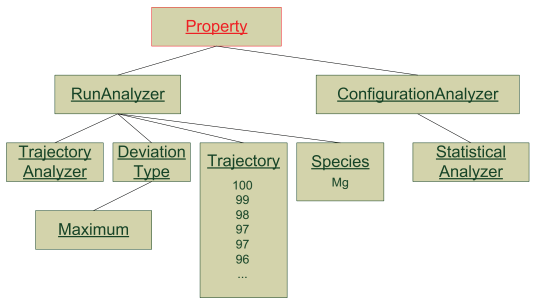

Figure 6: ParameterBlock for a property to get the maximum deviation between a given trajectory and the simulation results.

Mg	min deviation	0.0
	max deviation	11.4
	average deviation	-0.16
	average absolute deviation	2.63
Mg^+	min deviation	0.0
	max deviation	10.26
	average deviation	-0.15
	average absolute deviation	2.35
Mg^{2+}	min deviation	0.0
	max deviation	10.63
	average deviation	-0.003
	average absolute deviation	2.55

Table 1: Deviations between the trajectories of the Mg particles, produced by JAMES II and SPiM. All the deviations are averaged over the count of replications

tion of the model as well as the interpretation of the analysis results completely to the user, who would have to decide whether the numbers are satisfying or not. For all species the minimum deviation is 0.0, which denotes, that parts of the trajectories of JAMES II and SPiM overlap. The maximum deviation is between 10 and 12. The distance between the minimum and maximum deviation can be explained by the stochasticity of the models. Time points might exist, where the JAMES II trajectory has reached a maximum and the SPiM trajectory has reached a minimum (and vice versa), due to the use of different random numbers. Further experiments with equal random number generators and seeds (which would require including them into the ParameterBlock), would give more insight. Average and average absolute deviation denote, that over the whole simulation runs, the distance of the trajectories is small (less than 3 percent of the average species counts). However, as mentioned before, the decision, whether the results are satisfying, is left to the user.

5.3 Sensitivity Analysis

Sensitivity analysis is a major topic of validation. Experiments of this kind try to investigate the impact a change in the parameters of the model has on its behaviour. So far, we realized two variants of sensitivity analysis: sequential bifurcation and regression analysis with (fractional-) factorial experiment designs [25].

Sequential Bifurcation.

The basic idea of a bifurcation experiment is to group the parameters, according to whether a change of their value has an impact on the behaviour of the model. Therefore, interesting levels of the value domains of the parameters (in most cases two: high and low) are determined. The process of grouping the parameters is done iteratively. In the first step the parameters are divided arbitrarily into two equally sized groups. For both groups simulation runs are executed, with the high and low levels of the parameters. If there is no difference between the levels, the parameters of the accordant group are declared as insignificant. Otherwise, the group is divided again. This scheme is repeated until there are only single significant parameters, which form the result of the experiment.

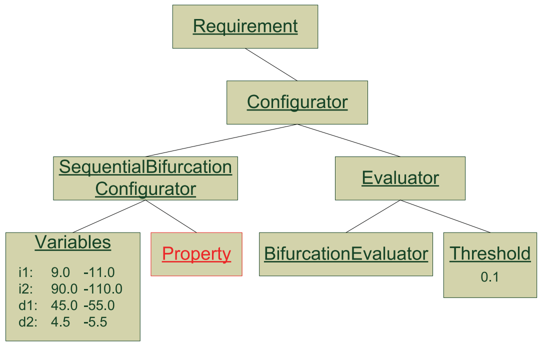

Figure 7: ParameterBlock for the sequential bifurcation experiment.

We use this kind of experiment, to illustrate the interaction between Configurator and Evaluator. The ParameterBlock to configure the *SequentialBifurcationConfigurator* contains a $HashMap$ mapping the parameters to their levels. For the example experiment we decided to set these levels to 10 percent below (low) and 10 percent above (high) the value used in the SPiM model description for each parameter. We used the same properties as in the simple validation experiment.

The Configurator holds a list of interesting parameter groups (which initially contains one group comprising all parameters). During an iteration step it removes the first group from the list, splits it into two equally sized groups and creates four VariablesAssignments (for each group low and high parameter settings). After simulation runs and analysis have been executed, the *SequentialBifurcationEvaluator* identifies the results of associated VariablesAssignments (two VariablesAssignments are associated, if they contain the same parameters but different assignments) and checks whether their distance exceeds a given threshold. If that is the case, the parameters are sent to the Configurator as feedback. The Configurator adds the parameters as a group in its list, to create new VariablesAssignments in a further step. It terminates the creation of VariablesAssignments, when the list is empty.

The threshold for the Evaluator has been set to 1 percent, which means that if one of the results of the deviations methods is more than 1 percent lower or higher than the corresponding result of the associated assignment, the group of parameters is marked to be significant. During the experiment with this threshold all four parameters of the

$MgCl_2$ were marked to be significant. Of course, this result highly depends on the threshold and on the used analysis method.

Factorial Experiment Designs.

The third experiment type we realized with the validation architecture is a sensitivity analysis, based on a factorial experiment design. The idea of a factorial design is again, to divide the value domains of the parameters (called *factors* in this context) into two (or more) levels. Based on these levels a full-factorial design can be created, where all possible factor combinations using the two levels are executed (similar to a boolean value table). Depending on the count of factors, the number of factor combination can be very high, due to the combinatorial explosion. However, under the assumption, that some factor combinations have a negligible impact on the simulation result, this number can be reduced. The calculation of such reduced factor combination sets is not trivial and an scientific field of its own deals with this problem [10]. An experiment design based on a reduced set, is called a fractional-factorial design.

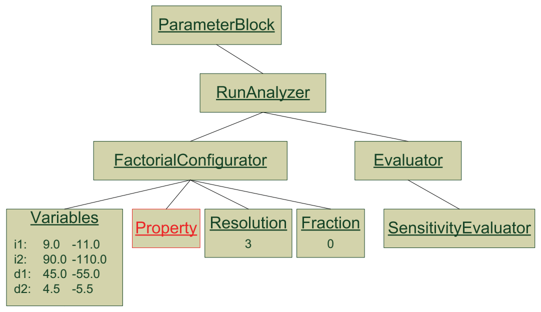

Figure 8: ParameterBlock for the factorial experiment.

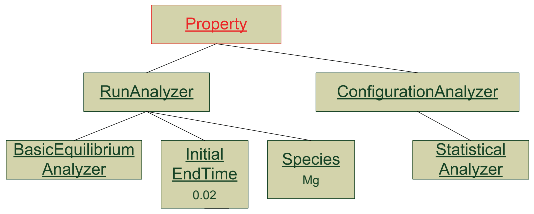

Figure 9: ParameterBlock for a property to get the equilibrium of a species.

We implemented the *FactorialConfigurator* to create such designs. Besides a similar *HashMap* as in the bifurcation experiment, the ParameterBlock comprises, two additional parameters, the fraction and the resolution. The fraction denotes the size of the subset of the full-factorial design, the resolution denotes the degree of factor combinations that are negligible. Based on these two parameters, a *FactorialDesignGenerator* is chosen, that calculates the design. The FactorialDesignGenerator has been relized as a plug-in type, to cover the variety of existing methods. So far we implemented three variants, a full-factorial generator (which we used during the experiments), an implementation according to the Franklin-Bailey method [18], and one according to Plackett-Burman [33]. Based on the design, the Configurator creates the VariablesAssignments.

The corresponding Evaluator calculates the sensitivity of the parameters based on the simulation results, by applying a regression analysis as described in [24].

In addition to the properties used in the previous experiments, we used different analysis method for our experiments with the factorial designs (see Figure 9 for the ParameterBlock). Therefore, we implemented a RunAnalyzer retrieving the steady-state of a simulation trajectory. Both properties were interchangeable in the overall ParameterBlock, representing the requirement, which illustrates the simple way, how the different components and methods can be combined. During all the experiment runs (with both properties), the model turned out to be insensitive to any of the parameters, which is interesting keeping in mind the result of the bifurcation experiment. This gives evidence, that the result of a validation experiment highly depends on the chosen methods and method parameters. In addition, it underlines the urgency for the preparation of experiments with proper definitions of requirements, while supporting the experiment execution by offering a broad variety of methods.

5.4 Face Validation with Mosan

During all of the experiments, we added the parameters for an additional evaluator to the requirements. Thereby, we enabled the evaluation of the simulation results with the *MosanEvaluator*, an Evaluator component wrapping the functionality of Mosan [39]. Mosan is a visualization tool for experiment results, in the context of systems biology. Figure

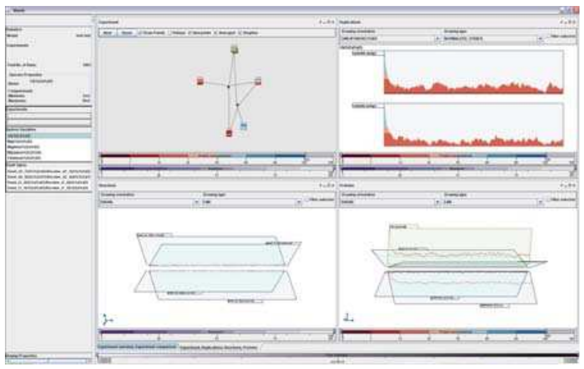

Figure 10: The result of an evaluation with Mosan.

10 shows an example for a Mosan visualization. In the upper left corner, a graph is shown representing the model. In the lower part of the model an overview over the trajectories of events and species is given. In the upper right corner more details are given about a specific trajectory. User interaction in one of the views, is reflected in the other views as well. For instance, if a user selects a species in the graph, the according trajectory is highlighted in the trajectory overview, which facilitates the browsing through the experiment results. The additional evaluation strategy is possible due to the high flexibility of the validation architecture, with respect to methods as well as to configurability. Furthermore, the opportunity to execute different evaluation strategies for the same set of configurations is exploited, by realizing a vi-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8833
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8833

sual as well as an automated evaluation of the experiments.

6. RELATED WORK

BIOCHAM [17] is a framework for the modeling and analysis of biochemical systems. It supports the simulation of differential equations, querying in temporal logic, and optimization methods for the model parameters. These three techniques cover the validation tasks: configuration, simulation, analysis, and evaluation. Requirements are defined in temporal logics. Since the techniques are tightly coupled and the simulation runs produce exactly the output the analysis step requires, no additional effort has to be put into the configuration of the observation. However, this tight coupling reduces the flexibility. While BIOCHAM is well suited to execute simulation-based model-checking experiments, it does not support the full spectrum of validation techniques (e.g., factorial experiment designs).

In [22] the authors describe a system for the automatic output analysis of discrete event simulations, including the estimation of required replications and simulation end times. Therefore, they propose an analyzer, sharing similarities, with the BaseAnalyzer and the associated components. At the beginning of the analysis process (and before the execution), the user determines whether a set of replicated simulation runs or one single long run shall be executed. In both cases, the user is then asked whether the warm-up period in the simulation output data has to be identified, which is done by the Warm-up Analyzer. Afterward, depending on the case, the count of required replications, or respectively the run duration is determined by suitable tools. The whole structure resembles the idea of dividing the analysis of a single run and a set of runs as we realized by separating the RunAnalyzer and the ConfigurationAnalyzer. However, the separation is realized very strictly, since the dynamic adaptation of run length and replication count cannot be combined. While this strategy may be appropriate for models, where the simulation runs reach a steady-state, it might not be useful for models with a less predictable behaviour. For instance events that occur rather late during the simulation run, but under specific conditions which highly depend on the initial stochastic behaviour of the model, could only be detected by adapting simulation end times and replication counts. Here, it could be necessary to have long simulation runs (to trigger the event) and many replications (to execute runs were the conditions are met). All three strategies (one long run, many replications, or a combination of both) can be realized with the BaseAnalyzer component of FAMVal.

SWAN Tools [30] offers a five-step workflow to guide the user through a simulation experiment. The first step is the configuration of the simulation parameters, e.g., RNG seeds and simulator components. During the second step, the model parameters are defined. The third step executes the simulation runs, based on interesting configurations and required replications. The fourth and fifth step include the analysis of the simulation results and the plotting of them. This scheme lays special focus on the configuration of model and simulation. However, the experiment is executed with just one simulator setting, which could bias the results. Furthermore, it is not possible to adapt the simulation end time, if the analysis process needs more data (e.g., when the warm-up phase of the simulation run is not finished).

7. CONCLUSION AND OUTLOOK

We introduced FAMVal, an architecture for experimental model validation. It is based on the six tasks of a validation experiment, specification of requirements, configuration of the model, model execution, observation, analysis, and evaluation. FAMVal tackles the emerging need of supporting modelers during the validation step(s) of the modeling and simulation process. The architecture comprises the components BaseValidator, ConfigurationSetValidator, BaseAnalyzer, and ReplicationAnalyzer to coordinate the experiment as well as the components Configurator, Evaluator, RunAnalyzer, and ConfigurationAnalyzer to provide the methods and techniques required for a validation experiment. The latter have been realized as JAMES II plug-in types to maximize flexibility and extensibility of the architecture and are coupled to the architecture by interfaces to avoid constraining the possible functionality. The definition of validation requirements in connection with the configuration of the experiment has been realized as ParameterBlocks.

Furthermore, we described example experiments, including simple validation experiments, sensitivity analysis, as well as face validation, to illustrate the power of the presented approach.

Future work will deal with adding new functionality by implementing additional plug-ins. Therefore, we are working on a RunAnalyzer checking for LTL-formulae in simulation trajectories and a ConfigurationAnalyzer, computing the probabilities that an LTL-formula is true for a set of (replicated) simulation runs. Both components are used to allow a (probabilistic) simulation-based model-checking. Furthermore, a ConfigurationAnalyzer is being developed that calculates the monte-carlo variability of a model. This component is based on methods from Bioinformatics [13] and social sciences [11] to retrieve the distance of strings adapted to compare simulation trajectories. To explore the parameter space of a model Configurators and Evaluators will be implemented, inspired by optimization methods.

Besides additional validation techniques and methods, further improvements are required. While ParameterBlocks are sufficiently flexible and adaptable to hold information about diverse validation requirements and to support a corresponding automatic configuration of experiments, they might not be the most convenient way to formalize the requirements for a user. Relief could be provided by a language to specify requirements in. A specific challenge for this language is that it should be adaptable to the specific level of abstraction in which the user would like to formalize his or her requirements for validation. In addition, the translation to detailed ParameterBlocks independently of the chosen abstraction level should be possible.

8. ACKNOWLEDGMENTS

This research is sponsored by the German research foundation (DFG).

9. REFERENCES

[1] ns-3 reference manual. Technical report, ns-3 Project, http://www.nsnam.org/docs/release/manual.pdf, 2009.
[2] O. Balci. Verification validation and accreditation of simulation models. In *WSC*, pages 135–141, 1997.

[3] O. Balci. Verification, validation, and certification of modeling and simulation applications. In *WSC*, pages 150–158, 2003.

[4] O. Balci and R. G. Sargent. A methodology for cost-risk analysis in the statistical validation of simulation models. *Commun. ACM*, 24(4):190–197, 1981.

[5] G. Batt, J. T. Bradley, R. Ewald, F. Fages, H. Hermans, J. Hillston, P. Kemper, A. Martens, P. Mosterman, F. Nielson, O. Sokolsky, and A. M. Uhrmacher. Working groups' report: The challenge of combining simulation and verification. In *Dagstuhl Seminar Proc. 06161: Simulation and Verification of Dynamic Systems*, 2006.

[6] S. Bensalem and D. A. Peled, editors. *Runtime Verification*. Springer, 2009.

[7] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 1976.

[8] D. Brade. *A Generalized Process for the Verification and Validation of Models and Simulation Results*. PhD thesis, Universität der Bundeswehr München, 2004.

[9] R. Butler, A. Geser, J. Maddalon, and C. Munoz. Formal analysis of air traffic management systems: The case of conflict resolution and recovery. In *WSC*, pages 906–914, 2003.

[10] J. Chen, D. X. Sun, and C. F. J. Wu. A catalogue of two-level and three-level fractional factorial designs with small runs. *International Statistical Review / Revue Internationale de Statistique*, 61:131–145, 1993.

[11] W. Dijkstra and T. Taris. Measuring the agreement between sequences. *Sociological Methods Research*, 24:214–231, 1995.

[12] R. Donaldson and D. Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In *CMSB*, pages 269–287, 2008.

[13] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[14] B. Edmonds and D. Hales. Replication, replication and replication: Some hard lessons from model alignment. *J. Artificial Societies and Social Simulation*, 6(4), 2003.

[15] K. H. Elster. *Modern Mathematical Methods of Optimization*. Wiley VCH, 1993.

[16] F. Fages and A. Rizk. On the analysis of numerical data time series in temporal logic. In *CMSB*, pages 48–63, 2007.

[17] F. Fages and S. Soliman. Formal cell biology in biocham (tutorial). In *Formal Methods for Computational Systems Biology*, pages 54–80, 2008.

[18] M. F. Franklin and R. A. Bailey. Selecting defining contrasts and confounded effects in p^{n-m} experiments. *Technometrics*, 27:165–172, 1977.

[19] J. M. Galan and L. R. Izquierdo. Appearances can be deceiving: Lessons learned re-implementing axelrod's 'evolutionary approach to norms'. *Journal of Artificial Societies and Social Simulation*, 8(3), 2005.

[20] J. Himmelspach, R. Ewald, and A. M. Uhrmacher. A flexible and scalable experimentation layer for JAMES II. In *WSC*, pages 827–835, 2008.

[21] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *Spring Simulation Multiconference*, pages 137–143. IEEE Computer Society, March 2007.

[22] K. Hoad, S. Robinson, and R. Davies. Automating discrete event simulation output analysis - automatic estimation of number of replications, warm-up period and run length. In *Simulation Society Workshop*, 2009.

[23] L. R. Izquierdo and J. G. Polhill. Is your model susceptible to floating-point errors? *Journal of Artificial Societies and Social Simulation*, 9(4):4, 2006.

[24] J. P. Kleijnen. Simulation experiments in practice: statistical design and regression analysis. *Journal of Simulation*, 2:19–27, 2008.

[25] J. P. C. Kleijnen. Design of experiments: Overview. In *WSC*, pages 479–488, 2008.

[26] J. P. C. Kleijnen. Kriging metamodeling in simulation: A review. *European Journal of Operational Research*, 192(3):707–716, 2009.

[27] S. Leye, J. Himmelspach, and A. M. Uhrmacher. A discussion on experimental model validation. In *UKSIM*, pages 161–167, 2009.

[28] G. H. Orcutt, S. Caldwell, I. W. Richard, S. Franklin, G. Henrricks, G. Peabody, J. Smith, and S. Zedlewski. *Policy exploration through microanalytic simulation*. Rowman & Littlefield, 1976.

[29] L. F. Perrone, C. Cicconetti, and G. S. andBryan C. Ward. On the automation of computer network simulators. In *SIMUTools*, 2009.

[30] L. F. Perrone, C. J. Kenna, and B. C. Ward. Enhancing the credibility of wireless network simulations with experiment automation. In *Workshop on Selected Topics in Mobile and Wireless Computing*, pages 631–637, 2008.

[31] A. Phillips and L. Cardelli. A correct abstract machine for the stochastic pi-calculus. *Electronic Notes in Theoretical Computer Science*, 2004.

[32] A. Phillips and L. Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In *CMSB*, pages 184–199. Springer, 2007.

[33] R. L. Plackett and J. P. Burman. The design of optimum multifactorial experiments. *Biometrika*, 33:305–25, 1946.

[34] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes - The art of scientifc computing*. Cambridge University Press, 3rd edition, 2007.

[35] T. J. Santner, B. J. Williams, and W. I. Notz. *Design and Analysis of Computer Experiments*. Springer, 2003.

[36] R. G. Sargent. Validation of simulation models. In *WSC*, pages 497–503, 1979.

[37] R. G. Sargent. Verification and validation of simulation models. In *WSC*, pages 157–169, 2008.

[38] R. E. Shannon. Tests for the verification and validation of computer simulation models. In *WSC*, pages 573–577, 1981.

[39] A. Unger and H. Schumann. Visual support for the understanding of simulation processes. In *Proceedings of IEEE Pacific Visualization Symposium*, 2009.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8833
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8833