ESS: EMF-BASED SIMULATION SPECIFICATION, A DOMAIN-SPECIFIC LANGUAGE FOR MODEL VALIDATION EXPERIMENTS

Joost Mertens
Joachim Denil

Faculty of Applied Engineering: Electronics and ICT
Flanders Make @ University of Antwerp
University of Antwerp
Groenenborgerlaan 171
Antwerpen, BELGIUM
{joost.mertens,joachim.denil}@uantwerpen.be

ABSTRACT

In the modeling and simulation process of systems, the verification and validation activities are necessary to give credibility to model results. For similar types of models, the applicable validation techniques are reusable and automatable. In this paper, we present ESS, the EMF-Based Simulation Specification. ESS is a domain-specific language, workflow, and accompanying code generators built on top of the Eclipse Modeling Framework. Existing approaches focus on specifying how experiments and simulations must be executed, ESS extends that functionality with the ability to specify and reuse model validation techniques. We provide an initial implementation of ESS that works with ordinary differential equations in Matlab and demonstrate this implementation with the validation of an electronic filter circuit. Though this initial version is limited to only one simulator and one validation metric, our aim is to provide support for an exhaustive list of validation metrics in the future.

Keywords: Validation and Verification, Domain Specific Language, Model-to-Text Transformation, Eclipse Modeling Framework.

1 INTRODUCTION

In the field of modeling and simulation (M&S), models are developed to answer questions about a system within a specific domain of application. Various flavors of the M&S process have been detailed in literature (Sargent 2020, Zeigler, Muzy, and Kofman 2018,Schlessingner et al. 1979). All these process models share some common activities that we summarize as follows: first, a system is analyzed, and the intended application domain of the model is formulated. Through this analysis, a conceptual model of the system is modeled, which excludes/abstracts away details of the system not pertaining to the intended application domain. Through programming, the conceptual model is transformed into a computer-executable model, which is simulated to answer the initial questions about the system. This process additionally involves three types of evaluation that ensure a credible model is obtained: (a) model qualification, where engineers check the adequacy of the conceptual model in the domain of application; (b) model verification, where engineers check that the conceptual model is correctly represented by the computer model; (c) model validation,

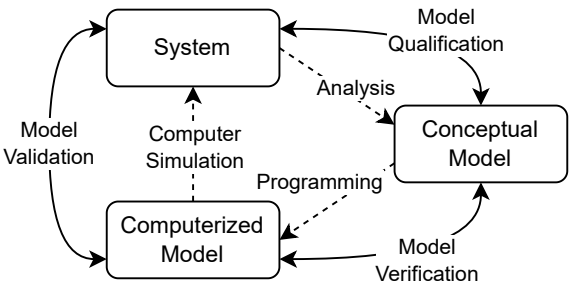


Figure 1: Modeling and Simulation phases, and evaluation procedures, adapted from (Schlessingner et al. 1979).

where engineers check that the simulation adequately represents the system within the computerized model's domain. This process is typically represented by the diagram shown in Figure 1.

In the model qualification of the conceptual model, engineers typically use face validation. The domain experts check the equations and their relations and ensure they are correct (Sargent 2020). In computerized model verification, we note two main parts: ensuring the conceptual model is implemented correctly and ensuring it executes correctly. Engineers attain the former through good coding practices, and for the latter, they generally rely on the owner of the simulator to provide a correctly working simulation engine (e.g. Mathworks ODE solvers, GNU Octave ODE solvers) (Sargent 2020, Oberkampf and Roy 2010). Blind reliance is questionable, certainly in high-stakes projects, but the large companies or communities behind those solvers allow us to trust them. Lastly, in model validation engineers compare real experiments with in silico experiments to assess the simulation results and the adequacy of the computerized model (Sargent 2020, Oberkampf and Roy 2010). When real experiments are impossible, they can instead compare with previously validated models. Model validation itself is commonly done with programmed code, which should also be subjected to verification. The same techniques from model verification can be applied. Although the techniques for model validation are generic for models of the same type, their application tends to be more ad hoc for each specific model. Because the techniques are generic, the potential for automation exists. This improves reuse and reduces manual effort.

In this paper, we present a domain-specific language (DSL) for specifying models and the validation of their results. The DSL is contained in a base package, which must be extended for specific models. We provide such an extended package for the case of Ordinary Differential Equation (ODE) models executed in Matlab, which we use in a demonstration on a model of an electronics circuit. Lastly, we present a workflow with model-to-code transformations for using the DSL in the validation of simulation models. We implement the DSL using the Eclipse Modeling Framework (EMF) (Steinberg, Budinsky, Merks, and Paternostro 2008), and utilize Eclipse Epsilon to facilitate code generation. As such, we call the DSL ESS: EMF-Based Simulation Specification. Besides the automation and technique reuse introduced with the model-to-text transformations, the DSL models additionally serve as a documentation store for the experiments performed during model validation. In the remainder of this paper, we discuss some background on model validation in Section 2, afterwards we discuss ESS in Section 3, then demonstrate it on the electronics circuit in Section 4. Finally, Section 5 discusses the relation with the related work and Section 6 summarizes the paper.

2 BACKGROUND

2.1 Model Validation

Model validation is defined in (Schlessingner et al. 1979) as: "Substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended

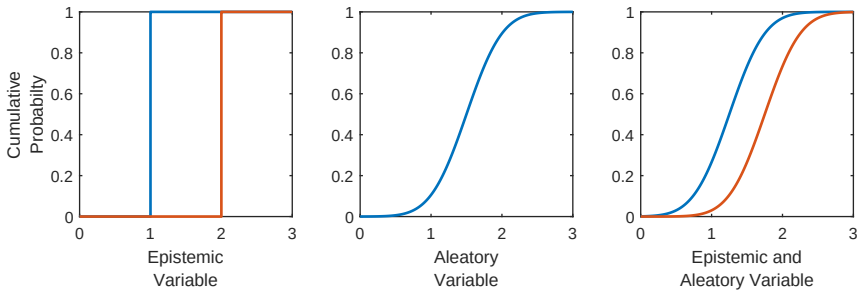


Figure 2: P-boxes of a variable with purely epistemic, purely aleatory, and combination of epistemic and aleatory uncertainty.

application of the model.". Other, similar definitions exist (Oberkampf and Roy 2010) but all approximately describe the following elements and processes: a system has certain properties that are of interest, within a domain of applicability. In literature these are called System Response Quantities (SRQ) (Oberkampf and Roy 2010), Quantities of Interest (QOI) (National Research Council 2012), or model outputs (Sargent 2020). The SRQ or QOI interpretation is broader than the model output interpretation, as they may be a post-processing of the model output. We can compare the model SRQs with experimentally obtained values of those SRQs, either through physical experiments or by simulation of previously validated models. Doing so enables us to quantify the accuracy of the model for the experiment domain. We can then, through extrapolation or interpolation, gather model results for the intended application domain, after which we can determine if the estimated accuracy of those results is within the specified accuracy requirements (Oberkampf and Roy 2010), also called tolerance (National Research Council 2012).

The SRQ of a system is dependent on the type of system. If we broadly categorize systems in discrete event and continuous systems (possibly with discretized time), we note that for the discrete event systems, SRQs are quantities like the throughput or the blocking rate of the system, whereas for continuous systems, we may be interested in steady-state values or transient quantities like settling time (Zeigler, Muzy, and Kofman 2018). For estimating the model accuracy, one needs a metric to be calculated over the simulation and the experimental data. This metric is called a validation metric operator (Oberkampf and Roy 2010) and is some kind of statistical operator, e.g., the difference between the means of the experimental and simulation SRQs. Just like the SRQs, it may depend on the type of system.

SRQs used for the calculation of the validation metric are usually not deterministic but contain uncertainty. There are two types of uncertainty: aleatory, where the uncertainty comes from the nature of random variables, e.g., measurement errors, and epistemic, where the uncertainty stems from a lack of knowledge. Aleatory uncertainties are represented as statistical distributions. The representation of epistemic uncertainties depends on how much knowledge is lacking. On the one end, if one lacks the knowledge to be aware of the epistemic uncertainty it can't be represented. On the other end, if one only lacks the knowledge of the statistical distribution, it can be represented by a bounded interval. Figure 2 illustrates three realizations of cumulative probability distributions (CDF) for a variable with epistemic, aleatory, and combined uncertainties. In the cases with epistemic uncertainty, there is a "min" and "max" CDF, for which the term probability box or p-box is used. To further clarify with an example: an electronic component whose parameter is given by its datasheet with a "min, max, typical" value has epistemic uncertainty, its realizations are contained in the p-box with two vertical edges as no knowledge of the underlying distribution exists. Measuring a batch of those components while disregarding the accuracy of the measuring device allows for the quantification to an aleatory uncertainty. Additionally incorporating the measuring instrument's accuracy allows for the quantification to an aleatory and epistemic p-box.

Validating a model requires awareness of the different uncertainties. For example, the model inputs often cannot be measured exactly, the model may not correspond 1-to-1 to the real system, and computerized simulation of the model execution may introduce solution errors. Various techniques exist to handle the uncertainties: Monte-Carlo simulation can be used to propagate aleatory and epistemic inputs or parameters through the model, obtaining a p-box of the output values of the model. Statistical Design of Experiments (DOE) techniques search to quantify different sources of uncertainty that contribute to the final uncertainty in the SRQ. Through thoughtful definition of the experiments, those specific sources can be isolated and quantified. Blocking is such a technique, in which experiments are grouped per potential source of uncertainty (Oberkampf and Roy 2010). Techniques used in model verification allow experimenters to quantify or eliminate solution errors (National Research Council 2012).

The previous remark on the p-boxes as representations of the uncertainty of model outputs brings along another complication: common validation metrics are based on classical statistical operators such as mean and standard deviation. Therefore, they can only handle aleatory uncertainty. Ferson and Oberkampf (2009), identify this problem and propose validation metrics that can deal with epistemic, aleatory and combined uncertainty. Of the proposed metrics, the most generally applicable one is called the "CDF area metric" in which the area is computed between the obtained p-boxes of the simulated SRQ and the experimentally measured SRQ. The main drawback of the CDF area metric is that its result requires interpretation of a domain expert to see if the value is adequate, and thus relies on human opinion. In contrast, most statistical tests can be interpreted by a non-expert, e.g., a hypothesis test at a certain confidence level. Ferson and Oberkampf (2009) additionally introduce a two-valued metric that is specifically applicable to epistemic observations, as it shows the width between the experimental and simulation bounds. Oberkampf and Ferson (2007) also present an approach called U-pooling, which allows low numbers of experimental observations to be pooled before applying the area metric, in an attempt to obtain a general model mismatch metric, rather than for each specific SRQ. For an overview of other validation metrics, we refer to (Liu et al. 2011).

3 ESS

We first discuss the ESS metamodel, then we discuss the workflow, and lastly, we cover the implementation.

3.1 ESS Metamodel

Figure 3 shows the metamodel of the ESS "core". We omitted some blocks, attributes, and associations/compositions to maintain readability, however, the essence of the metamodel remains. The core of ESS consists of 5 main classes: Model, Scenario, Experiment, SystemResponseQuantity, and ResultAnalysis. Optionally encapsulating these classes is the System class (not shown in Figure 3). The core itself does not implement all the necessary information for performing model-to-text generation, therefore most of the classes are abstract, and need to be extended in a simulator-specific metamodel, which we cover later on. In what follows, we discuss the metamodel elements from Figure 3, which are italicized for clarity.

The *Model* class is an abstract representation of a computerized model, which contains arbitrary many *Ports*. A port is the abstracted class through which a model communicates, example implementations could be a model input port, output port, or initial state. An abstract model may still be too generic and can therefore be extended by still abstract but more specific models, shown in Figure 3 by the *ODEModel*. Such an extension adds non-implementation-specific attributes, while still being part of the core.

The *Scenario* class describes a specific system setup and scenario. It does so by specifying various attributes or parameters of the system and its surroundings. We call those values *SystemValues*.

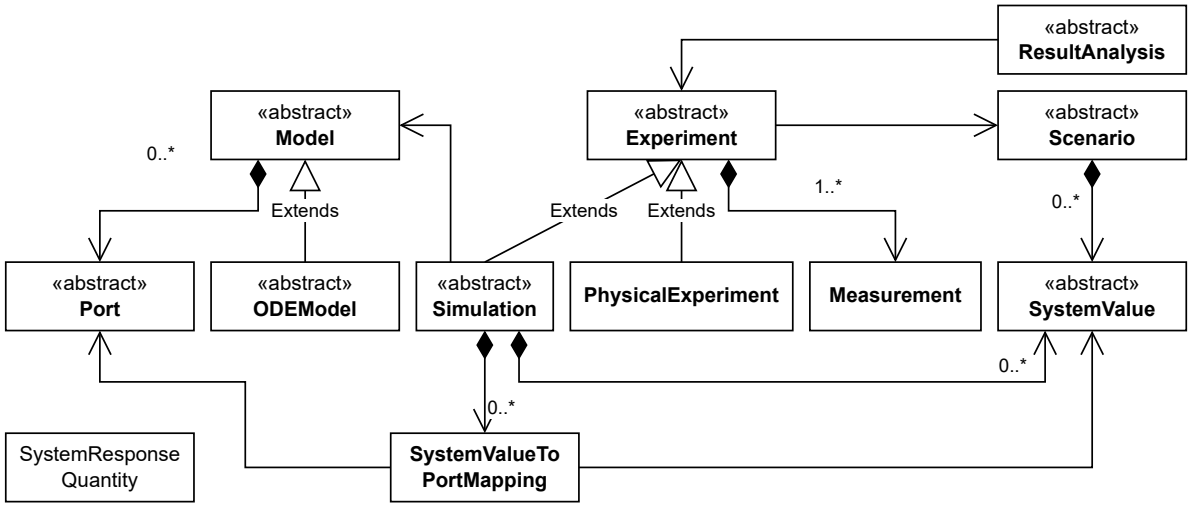


Figure 3: ESS "core" metamodel, with omissions for readability.

The *Experiment* class formulates an experiment, following a *Scenario*, and producing one or more *Measurements*. There are two specializations of the *Experiment*, namely the *PhysicalExperiment* and the *Simulation*. The *PhysicalExperiment*'s purpose is to describe an experiment that requires real-life execution. The *Simulation* pertains to an in silico experiment, and therefore it's associated with a *Model*. Through one or more *SystemValueToPortMappings*, the *SystemValues* of the *Scenario* are coupled to the *Ports* of the *Model*. Although a *Scenario* can define more *SystemValues* than needed for a *Simulation*, it's also possible that not all *Ports* of the *Simulation* are assigned a *SystemValue*, perhaps because the *Scenario* is lacking, but also possibly because the *Port* may not be related to any part of the real system captured in a scenario. For example, this can be the case if the model has tweakable parameters that are unrelated to the real system but only serve to improve the model response. In such a case, an engineer may additionally define *SystemValues* in the *Simulation* itself, and then map those to *Ports* of the *Model*.

The *ResultAnalysis* is associated with one or more *Experiments* and serves to define any post-processing needed on their *Measurements*. For example, a simple plot of a *Measurement* or, the calculation of a validation metric.

Lastly, there are the *SystemResponseQuantities* of the system. SRQs serve a purely descriptive role and are referenced by the other main elements to indicate that that SRQ is for example calculated in a *ResultAnalysis* or is contained in the *Measurement* of an *Experiment*.

3.2 ESS-Matlab ODE Metamodel

Code generation based on the ESS metamodel is impossible, as it lacks the necessary implementation-specific details to do so. Therefore, for each simulator, it is necessary to provide subclasses of the abstract classes in the core metamodel that do contain those details. For our case of validating Matlab ODE models, we extend the core metamodel with specific Matlab ODE classes. Containing the ESS model in a package and importing it in the Matlab metamodel makes it possible to extend and use its classes. Figure 4 shows this relation and the classes from ESS which are extended by the ESS-Matlab package. Given that the *PhysicalExperiment* is not abstract, we need only extend the remaining 4 classes (some more classes are extended, e.g. the *SystemValue* class, but are not shown for conciseness). Examples of the additions done in the Matlab specific classes are the specification of a solver that yields a valid result for the model, the specification of whether a *Simulation* needs reproducible random number generation, and the definition of

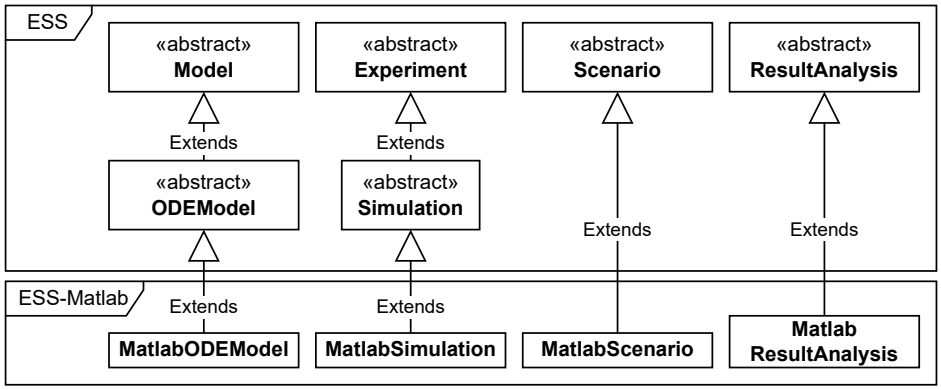


Figure 4: ESS-Maltab extensions of the ESS Package.

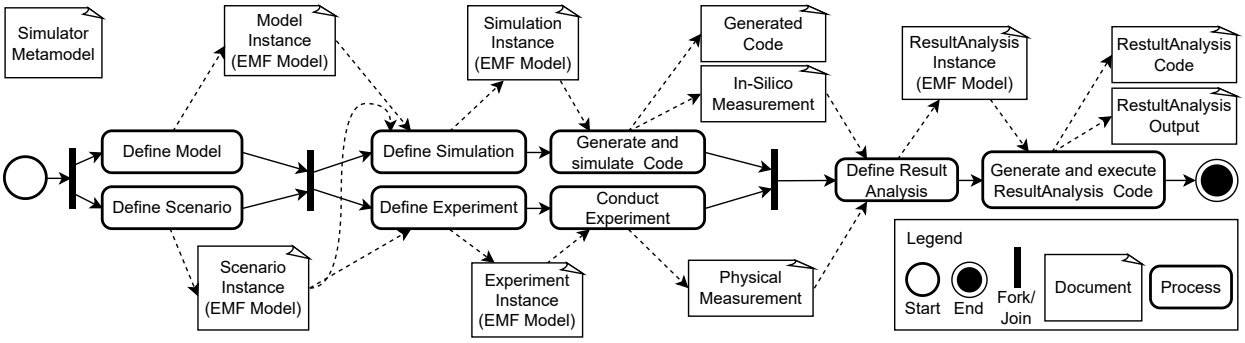


Figure 5: Workflow of using ESS. The metamodel serves as input for all of the "define X" processes.

some more specific *SystemValues* that adhere to specific probability distributions, the option for parallel the execution. The current limitations of the ESS-Matlab package are that models are limited to Matlab ODE models, and the result analysis is limited to the CDF area metric, calculated in Matlab.

3.3 Model-to-Code Transformations

We require two types of model-to-code transformations: (a) a transformation from a *Simulation* to code, to set up the in silico experiment and (b) a *ResultAnalysis* transformation to perform further post-processing and the calculation of the validation metric. The code resulting from those transformations implements the epistemic-aleatory simulation loop and CDF area metric from (Oberkampf and Roy 2010, Ferson and Oberkampf 2009). Other validation metrics can be added in future work.

3.4 Overview of Workflow

Lastly, we visually summarize the workflow and the input/output artifacts of the ESS approach in the activity diagram in Figure 5. A document is an artifact that follows from the process, manually for the definition of models, or automatically for the generation of code.

3.5 Implementation

ESS is built using the Eclipse Modeling Framework (EMF), EMF was conceived as a modeling framework and code generator that unifies XML documents, UML diagrams, and Java code through a high-level representation called the EMF model (Steinberg et al. 2008). Based on this EMF core, other frameworks are built for purposes such as model transformations and graphical user interfaces. For ESS we utilize the EMF core and the EMFatic language to define the metamodels of the DSL. For the model-to-text transformation, we use Eclipse Epsilon, which is a set of scripting languages for performing model-to-X transformations, model validation and testing, model merging and migration, and visualization on EMF models. For instantiating the EMF model objects, e.g., a *Simulation*, we rely on the built-in Exeed model editor. Exeed is straightforward tree-based editor.

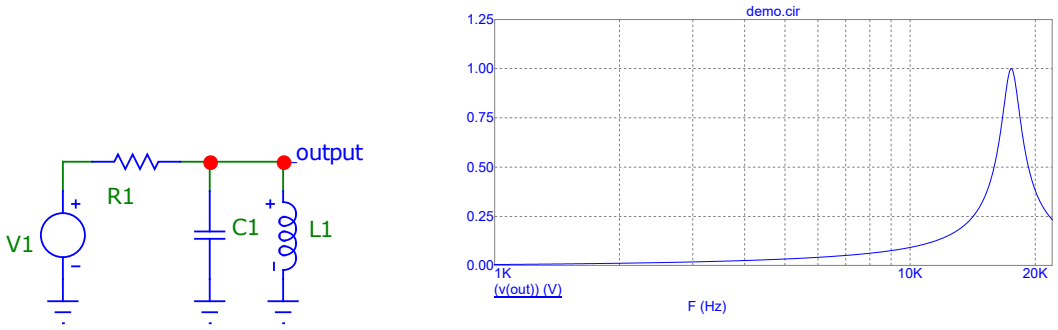
Specifically for the model-to-text transformations, we provide some more detail: we utilize the Epsilon Generation Language (EGL) and the EGL Coordination Language (EGX). EGL is a template-based model-to-text transformation language, which means the EGL script has the same layout as the text that is generated from it. EGX is an execution language that allows for the parametric execution of EGL scripts. An EGX script defines one or more rules that are invoked on EGL templates and adds additional functionality such as guards that prevent erroneous code generation. For our two model-to-text transformations, we create two EGL templates, the first is to be executed on a *MatlabSimulation* to generate a Matlab script that runs the simulations. The second is to be executed on a *MatlabResultAnalysis* that refers to a *MatlabSimulation* and a *PhysicalExperiment*, it generates code to calculate the CDF-area validation metric.

The choice for only the EMF core with Epsilon scripts was made as we deemed it to be the most lightweight approach to reach our goal of defining experiments, models and simulations, and accompanying transformations. In hindsight, this mostly code-based approach is not the most user-friendly for end-users. Nonetheless, more powerful tools build on EMF exist that would enable the creation of a full-fledged DSL workbench. Examples are Eclipse Sirius and Obeo Designer.

4 DEMONSTRATION ON AN ELECTRONICS CIRCUIT

We demonstrate the approach on an electronics circuit of an RLC notch filter. The idealized circuit of this filter is given in Figure 6a and its frequency response in Figure 6b. As seen in this response, the filter aims to suppress all frequencies besides those at the peak, also called its center frequency. The peak additionally has a width, called the bandwidth of the filter. At its peak the filter also has a certain amplitude. We choose those three parameters as SRQs for evaluating this system. This idealized model skims over various non-ideal effects, some of which should preferably be modeled as well to achieve a representative response. For this circuit, the most important ones are the output impedance of the function generator (R0), the DC resistance of the coil L1 (DCR) and the leakage current of the capacitor C1 (DCL). They affect both the amplitude and the bandwidth of the filter. R0 can be added to the circuit as a resistor in series with V1, DCR can be added as a resistor in series with L1, and DCL can be added to the circuit as a parallel resistor to C1. The circuit from Figure 6a was modeled as an ODE in Matlab, additionally incorporating R0 and DCR. DCL was not modeled as the we used a film capacitor with a low leakage current in the real circuit.

For creating the necessary EMF models, we follow the workflow from Figure 5. First, we define a *MatlabODEModel* and a *MatlabScenario*. The scenario defines the components we select for the filter. After analyzing the component datasheets and the capabilities of our measuring device, we determined that the 4 measurable components for the scenario all have epistemic uncertainty: R1 and C1 and the series resistance of L1 can be measured and are therefore epistemic uncertainties within the measurement accuracy of the measuring device. L1 cannot be measured and is therefore an epistemic uncertainty within the tolerances provided in its datasheet. With those two models created, we define the *Simulation* and *Experiment* models.



(a) Ideal circuit.

(b) Frequency response (magnitude).

Figure 6: RLC notch filter demo.

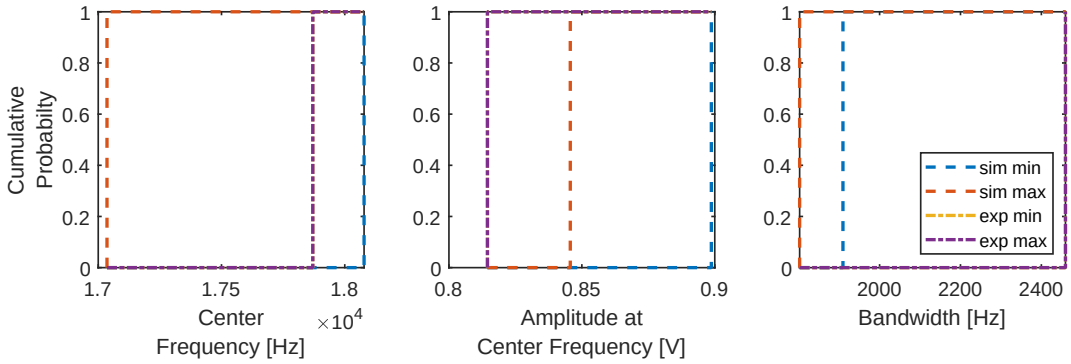


Figure 7: P-boxes for the three SRQs. Note: experiment min. and max. overlap due to only one observation.

In both the experiment and simulation, an exponential sine wave chirp signal is applied at the input of the circuit (V1), going from 10 Hz to 22 kHz over a duration of 10 s. Doing so for the experiment required constructing the circuit in real-life, connecting it to a function generator and measuring the sweep with an oscilloscope. In the simulation the measured sweep is the output of a simulation run. For the simulation, we pick several replications, in each of which the epistemic uncertainties are sampled randomly. Oberkampf and Roy (2010) provide a rule of thumb of $a^3 + 2$ replications if a is the number of epistemic uncertainties that are sampled according to Latin hypercube sampling. By conducting the experiment, we obtain one real-world measurement, and running the simulation yields one measurement for each replication. Lastly, we define a *MatlabResultAnalysis* for processing and comparing those simulation measurements and the experimental measurement. In this case the analysis entails some pre-processing before the area metric can be calculated: a conversion to the frequency domain using the Fast Fourier Transform, and the extraction of the three SRQs from the circuit response.

By running the analysis of the results, we obtain Figure 7, which contains 3 CDFs. Because the propagated input parameters are epistemic, the CDFs are p-boxes with vertical edges. Since the experiment, was only a single measurement, the min and max CDFs overlap each other. The *ResultAnalysis* returns the following CDF area metric values: Center Frequency = 0 Hz, Amplitude at Center Frequency = 0.0312 V and Bandwidth = 550.20 Hz. The interpretation of those results requires domain knowledge when the area is not 0. Therefore, from these results, we conclude that the model appropriately predicts the center frequency. Applying our domain knowledge, we can state that the 0.0312V amplitude area could be acceptable, but the 550.20 Hz bandwidth area is not. Therefore, if we want to use this model to predict the 3 SRQs, model refinement is needed.

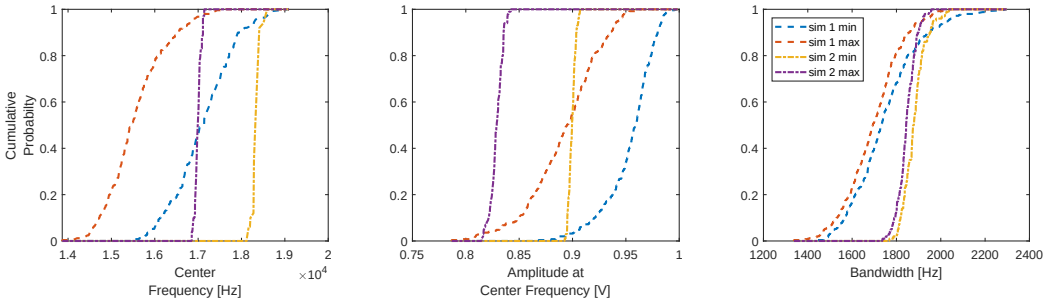


Figure 8: Demonstration of the p-boxes for results that are both aleatory and epistemic.

4.1 Extension with Aleatory Uncertainty

The demonstration on the real filter only has epistemic uncertainties, thus, we could not demonstrate that the implementation of the CDF calculation can handle the combination CDF of aleatory and epistemic uncertainties as introduced in Figure 2. To demonstrate this capability, we generate two hypothetical simulations, where there are additional aleatory input uncertainties present, and apply the area metric calculations to those results. Furthermore, due to the large number of replications, we utilize the parallel execution option of the *MatlabSimulation* instance to run the simulations. We made the probability distributions of the input parameters different, such that we obtain distinct p-boxes. For each simulation, we opt for 250 aleatory replications and 11 epistemic replications, totaling 2750 simulations that store approximately 50 GB of data. From this data, we produce the CDFs shown in Figure 8. The calculated areas are: Center Frequency = 207.4186 Hz, Amplitude at Center Frequency = 0.0086 V and Bandwidth = 113.6713 Hz.

5 RELATED WORK

5.1 SESSL, Validity Frames and Experimental Frames

Using model driven approaches to describe and set up simulations has been done previously in (Ewald and Uhrmacher 2014, D'Ambrogio, Bocciarelli, Delfa, and Kisdi 2020). D'Ambrogio et al. (2020) describe how EMF and related technologies can be used to implement a toolchain for large scale distributed simulations, as demonstrated on an ESA (European Space Agency) project. Ewald and Uhrmacher (2014) describe SESSL, which is an acronym for Simulation Experiment Specification via a Scala Layer. SESSL is an embedded DSL that is implemented in the Scala programming language and is used to specify simulation experiments (Ewald and Uhrmacher 2014). SESSL was conceived as a DSL to provide a system-independent way of describing simulation experiments. It stems from the problems with attempting to unambiguously describe a simulation experiment in a reproducible manner. An embedded DSL, in contrast with a regular DSL, is implemented in or "embedded" in a general-purpose programming language and can therefore be mixed and combined with that host language. To define an experiment in SESSL means to program that experiment, its execution is therefore unambiguously defined.

SESSL consists of abstract core functionality, in the form of the "AbstractExperiment" class, and accompanying abstract traits: "BasicExperimentConfiguration", "AbstractObservation", "AbstractParallelExecution", et cetera. To provide an implementation for a specific simulator, a binding needs to be written that through inheritance implements the needed functionality of the abstract trait from the SESSL core. In this way, the SESSL core remains generally inheritable, whilst the specific bindings need only implement the supported functionality of that simulator. SESSL currently has bindings for the following simulators: ML-Rules, ML3, SBMLsimulator, OMNeT++, pSSAlib, SSJ, Opt4J, SBW, and JAMES II.

Wilsdorf et al. (2019) present an extension on SESSL that based on an experiment schema helps users define specific experiments. Through a mapping of the schema to template fragments, the abstract experiment definition can be mapped to an implementation in SESSL (or another simulation backend). In essence, this is an additional layer of abstraction, which hides the SESSL experiment specification in Scala from the user and presents them with an experiment-type specific schema and graphical user interface. Thanks to the schema definition, validation of the user input data is also possible, notifying users of errors in their input.

The Experimental Frame (Zeigler, Muzy, and Kofman 2018) and Validity Frame (Denil et al. 2017) are also related to ESS. Zeigler (1976) introduced the experimental frame (EF) as "... a specification of the conditions under which the system is observed or experimented with.". Traoré and Muzy (2006) generalize the experimental frame to just the frame, as they identify that the wording of the EF allows for multiple interpretations. With this generalization, they introduce multiple views to which a frame may refer. Of those identified views, three are particularly applicable in the context of this paper, those are the views where a frame captures "The way a real system is observed...", "The way a model is experimented with...", and the "The circumstances under which a model is a valid representation of a real system...". The experimental frame, as defined by Zeigler, Muzy, and Kofman (2018) and Traoré and Muzy (2006) only specifies a family of experiments, but not a specific experiment, as such Denil et al. (2017) make note of the missing information needed to fully reproduce experiments with an EF and introduces the Experimental Setup, which does contain this information. They encapsulate the experimental setup in the Validity Frame, a frame that "defines the experimental context of a model in which it gives predictable results" (Denil, Klikovits, Mosterman, Vallecillo, and Vangheluwe 2017).

5.2 Discussion

SESSL and ESS originate from different ideas with different goals. For SESSL, the goal is unambiguously describing simulation experiments, for ESS the goal is facilitating the reuse of model validation experiments and their accompanying validation metric calculation. To do so, both require simulation descriptions, and as such, there is some overlap at that level, as both share a DSL with which experiments and related data can be defined. Architecturally, there is also some similarity, as both rely on the extension of the core to add support for simulators/types of experiments.

Without going into too much implementation detail, regarding the description of in silico experiments, we can say that in essence that part of ESS is similar to SESSL with the schema extension, minus the use of an intermediary simulation layer. ESS goes directly to the specific simulator, which means ESS employs one layer of abstraction less. This does require that developers of metamodels and model-to-text transformations are aware of the intricacies of the target simulator. In fact, the definition of the simulation experiment in the workflow could conceptually be replaced with a SESSL description of that experiment, something that we consider as future work. One implementation detail we do note is that for the experiment specification, ESS decouples the scenario and experiment, and requires the developer to either assign a scenario to a *PhysicalExperiment* for documentation purposes or assign and map elements of a scenario to model inputs for a *Simulation*. Because SESSL only focuses on simulation experiments such an approach is not needed, and the simulation scenario is inherent to the experiment configuration.

Regarding the calculation of validation metrics, and the processing of experimental measurements we can state the following: SESSL supports result processing of simulation results. However, because it only serves simulation experiments, this processing cannot perform model validation, for which a comparison with experimental measurements is needed. ESS does support this processing, as this is precisely its goal from the start. For performing the validation, in ESS developers have the advantage that they can write the processing in their language of preference, only needing to map it to an EGL template for further re-use.

On the topic of frames, Van Mierlo et al. (2020) explore Validity Frames in practice by demonstrating the Validity Frame vision on a model of an electrical resistor. They state how the validation procedure is an explicit activity stored within the Validity Frame and propose a validation procedure for the electrical resistor model. Because of this, we can envision the ESS approach as an embedded part of a Validity Frame, describing the validation activity, as well as providing the executable code to perform it. Model-to-model transformations would be necessary to obtain an ESS model from a model's frame without overhead.

Regarding the general use of ESS in the context of modeling and simulation, we recall that the main goal is on the topic of model validation. With the current support of only the CDF metric, and only through Matlab, this general applicability is limited, and as such, the main future task is to extend the supported types of result analyses with (a) more validation metrics, (b) support beyond Matlab, (c) user guidance as to which metrics to select. The (in silico) experiment description part is in this regard less interesting for future work, in part due to it existing in other solutions such as SESSL. Some final thoughts are that for future work on ESS there is a broad list of options available in the EMF ecosystem. There's Eclipse Sirius and Obeo designer for making graphical user interfaces, the option to use other model-to-text transformations such as Acceleo, and model validation with Eclipse EVL in Epsilon.

6 SUMMARY

We identified that for similar types of models the model validation techniques can be reused and automated. To this end we introduced ESS, a DSL, code generators and workflow built on top of the EMF framework. ESS allows developers to extend the ESS core and write packages and code generation templates to support a specific type of model and simulator, which we demonstrated for Matlab ODE models. We compared ESS to SESSL, a DSL made for unambiguously describing simulation experiments.

Regarding technical future work, we aim to extend the capabilities of ESS with respect to the number of validation metrics and the supported underlying languages/tools. Support for a large amount of validation metrics is a must to attain general applicability in a modeling and simulation context. Second to this main goal, we want to utilize the capabilities of EMF and Epsilon, to provide model validation of user models, completely automated tasks that implement the manual workflow, and create a graphical editor in place of the Exeed tree-based editor. A final future work is to develop a SESSL binding for Matlab ODEs, to attempt to replicate the simulation description step in SESSL as well.

Regarding research-oriented future work, we want to look into using the result analysis defined in ESS for the automatic detection of malfunctions or wear in real systems. We think that comparing system measurements, e.g., from a startup or calibration phase, with expected measurements made during the validation of the system model at design time, can help detect problems in a system.

ESS is available online: https://cosysgit.uantwerpen.be/JMertens/ess.

ACKNOWLEDGMENTS

Joost Mertens is funded by the Research Foundation - Flanders (FWO) through strategic basic research grant 1SD3421N.

REFERENCES

Denil, J., S. Klikovits, P. J. Mosterman, A. Vallecillo, and H. Vangheluwe. 2017. "The Experiment Model and Validity Frame in M&S". In *Proceedings of the Symposium on Theory of Modeling amp; Simulation*, TMS/DEVS '17. San Diego, CA, USA, Society for Computer Simulation International.

D'Ambrogio, A., P. Bocciarelli, J. Delfa, and A. Kisdi. 2020. "Application of a Model-Driven Approach to the Development of Distributed Simulations: the ESA HRAF Case". In *2020 Spring Simulation Conference (SpringSim)*, pp. 1–12.

Ewald, R., and A. M. Uhrmacher. 2014. "SESSL: A Domain-Specific Language for Simulation Experiments". *ACM Trans. Model. Comput. Simul.* vol. 24 (2).

Ferson, S., and W. L. Oberkampf. 2009. "Validation of imprecise probability models". *International Journal of Reliability and Safety* vol. 3 (1-3), pp. 3–22.

Liu, Y., W. Chen, P. Arendt, and H.-Z. Huang. 2011. "Toward a better understanding of model validation metrics". *Journal of Mechanical Design* vol. 133 (7).

National Research Council 2012. *Assessing the reliability of complex models: mathematical and statistical foundations of verification, validation, and uncertainty quantification*. National Academies Press.

Oberkampf, W. L., and S. Ferson. 2007. "Model Validation Under Both Aleatory and Epistemic Uncertainty". Technical report.

Oberkampf, W. L., and C. J. Roy. 2010. *Verification and validation in scientific computing*. Cambridge University Press.

Sargent, R. G. 2020. "Verification And Validation Of Simulation Models: An Advanced Tutorial". In *2020 Winter Simulation Conference (WSC)*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, pp. 16–29.

Schlessingner, S., R. E. Crosbie, R. E. Gagné, G. S. Innis, C. S. Lalwani, J. Loch, R. J. Sylvester, R. D. Wright, N. Kheir, and D. Bartos. 1979. "Terminology for model credibility". *SIMULATION* vol. 32 (3), pp. 103–104.

Steinberg, D., F. Budinsky, E. Merks, and M. Paternostro. 2008. *EMF: Eclipse Modeling Framework*. Second ed. The Eclipse Series. Addison-Wesley.

Traoré, M. K., and A. Muzy. 2006. "Capturing the dual relationship between simulation models and their context". *Simulation Modelling Practice and Theory* vol. 14 (2), pp. 126–142.

Van Mierlo, S., B. J. Oakes, B. Van Acker, R. Eslampanah, J. Denil, and H. Vangheluwe. 2020. "Exploring Validity Frames in Practice". In *Systems Modelling and Management*, edited by Ö. Babur, J. Denil, and B. Vogel-Heuser, pp. 131–148. Cham, Springer International Publishing.

Wilsdorf, P., M. Dombrowsky, A. M. Uhrmacher, J. Zimmermann, and U. v. Rienen. 2019. "Simulation Experiment Schemas –Beyond Tools and Simulation Approaches". In *2019 Winter Simulation Conference (WSC)*, edited by N. Mustafee, K.-H. G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, pp. 2783–2794.

Zeigler, B. P. 1976. *Theory of modeling and simulation*. Wiley-Interscience.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press.

AUTHOR BIOGRAPHIES

JOOST MERTENS is a PhD student at the University of Antwerp, Faculty of Applied Engineering in Electronics and ICT. His research interests lie in physics-based digital twins and model testing. His email address is joost.mertens@uantwerpen.be.

JOACHIM DENIL is an Assistant Professor at the University of Antwerp, Faculty of Applied Engineering in Electronics and ICT. His research interests include Multi-Paradigm Modeling and model-based systems engineering. His email address is joachim.denil@uantwerpen.be.