Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

A GENERAL FRAMEWORK FOR EXPERIMENTAL DESIGN, UNCERTAINTY QUANTIFICATION AND SENSITIVITY ANALYSIS OF COMPUTER SIMULATION MODELS

Sichao Wu

Network Dynamics and
Simulation Science Laboratory, VBI and
Department of Computer Science
Virginia Tech
1880 Pratt Drive
Blacksburg, VA 24061, USA

Henning S. Mortveit

Network Dynamics and
Simulation Science Laboratory, VBI and
Department of Mathematics
Virginia Tech
1880 Pratt Drive
Blacksburg, VA 24061, USA

ABSTRACT

Rigorous design of experiment (DOE) is essential to conduct validation, uncertainty quantification (UQ), and sensitivity analysis (SA) of computer simulation models. However, executing the process often involves knowledge of data management, statistical design, running simulation model, data analysis, and so on. It is a non-trivial task even for domain experts without solid computing backgrounds. Besides, the lack of standardization of data formats, configuration specifications, model invocation and execution mechanisms makes the process a harder undertaking. In this paper, we propose a comprehensive framework to support efficient experimental design, and UQ/SA in a domain and model independent manner. The data management and model execution issues are handled transparently from the users so that they can focus on the analysis itself. An application example is provided as an illustration of the concepts and basic use of this framework.

1 INTRODUCTION

1.1 Background and Related Work

Uncertainty quantification (UQ) and Sensitivity analysis (SA) are integral components in development of mathematical models and their implementations as large-scale computer simulations, see (Committee on Mathematical Foundations of Verification, Validation and Uncertainty Quantification et al. 2012) and references therein. However, the level of complexity involved in executing experimental design supporting these processes is high, and the process typically involves many scientific domains. Examples of challenges that a user would encounter include the following:

- Handling a multitude of formats for input data and configuration specifications.
- Being knowledgeable of statistical designs, in particular sparse designs for computer experiments.
- Executing complex computer simulation models on advanced computing platforms.
- Management of all the input and output data associated to each cell of the statistical design.
- Working with and efficiently analyzing the data produced.
- Allowing for principles, theory and solutions to be applied across different simulation models.

The framework presented here addresses all these points. It is aimed to provide a comprehensive environment for supporting efficient experimental design, model execution, data management, and UQ/SA analysis in a domain-independent manner.

Computer simulation models often involve a pipeline consisting of executables with mechanisms for invocation. Such pipelines can be a single model or can be composed of a collection of sub-models with data or communication dependencies among them, which we refer to as a computational workflow. Different simulation models, created by different users for different applications, often have extensive differences of formalization, structure, workflow, running environment, data management and analysis mechanisms. Since such large diversity exists, for general experiment practitioners, the challenge of conducting computer simulations lies in the fact that it usually requires extensive expertise in hardware, software, high performance computing (HPC), implementation, analysis, statistics, and the model itself. The lack of standardization of the whole process prevents general users from making use of the simulation models in an efficient way. However, at an abstract level, if a model has been constructed, the generic simulation procedure can be formulated as following steps: select an appropriate experimental design, execute the simulation model for the selected cells, interpret the results, validate the model or recommend future action as illustrated in Figure 1. On the one hand, simulation outcomes are essential data for UQ/SA studies which in turn provides guidance to calibrate and validate the model, etc. On the other hand, they can be used as feedback for experimental refinement, also known as adaptive design. Though the procedure is clear at an abstract level, it is generally hard to build a "universal" system that accommodates a broad range of models.

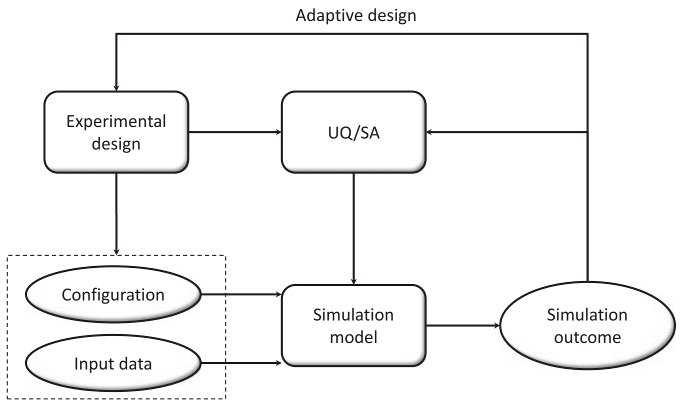

Figure 1: General abstraction of DOE and UQ/SA.

Due to the fact that computer experiments often have a large parameter space and running each single experiment instance is possibly quite time- and resource-intensive, traditional experimental design methodologies such as factorial design cannot meet the scalability requirement (Santner, Williams, and Notz 2003). During the last decades, methods such as Latin hypercube sampling (McKay, Beckman, and Conover 1979), orthogonal arrays (Hedayat, Sloane, and Stufken 1999) and other general sparse/space-filling designs (Johnson, Moore, and Ylvisaker 1990) are developed to reduce the number of simulation runs while still capturing the model variability. But, for a relatively long time, the experimental design process is conducted manually by the practitioner in an ad hoc manner. In recent years, researchers have developed systems automating the computer simulations. Though such systems vary in technical and implementation details, they share the common aspects in terms of motivation and goals - build systems that provide reliable, repeatable, fast, easy-to-use experimental design and analysis environment. For example, SAFE is a computer aided simulation tool that automates the experimentation for $ns-3$ networks (Perrone, Main, and Ward 2012). It provides user-friendly interfaces for both experienced power users and novice users. Another example is JAMES II (Ewald et al. 2010), which supports modeling, experiment design and execution, output data management, and data analysis in the domain of biology. However, these systems are designed for particular and quite specialized application domains or models. DAKOTA (Adams et al.

2009) is a software toolkit developed by Sandia National Laboratories. It is intended as an interface between simulation codes and a variety of analysis methods including UQ/SA. But it does not account for the big data management issues which is essential in modern computer simulations. Particular statistical softwares such as Minitab and SPSS in market are powerful tools for experimental design and analysis, but they do not have interfaces to couple the customized simulation code and handle the model execution and data issues. To the better of our knowledge, we are not aware of comprehensive systems supporting UQ/SA studies for large-scale and data-intensive computer simulation models as the framework we present here.

1.2 Contribution

Motivated by above-mentioned aspects, the goal of our work is to generalize the process of experimental design and UQ/SA across various domains, simulation models, and running platforms. To support this, we assume a comprehensive environment with a central registry that includes standardized data types, data classes, data sets and workflow specification along with their descriptive metadata. Our framework directly interacts with the registry for data, and model execution requests instead of with the physical infrastructures to ensure that our system is generic, broadly applicable and extensible. However, we do not assume particular form of the central registry systems but provide adapters accommodating them. By formulating standardized specifications for model configuration, workflow definition, input and output format, our design lets the users carry out experimental design, run simulation models, manage data, analyze the simulation outcomes through general interfaces even without expert knowledge about the simulation model they use. Since actually implementing and running the computer simulations are specific to the models and systems, we have scripts which is a part of the central registry to transform the standardized specifications to and from the model-native formats, which is performed automatically and hidden from the user at a low level (there may be other translation tools for large classes of models). We also provide tools for automatic experimental design, UQ/SA studies, and adaptive design. These tools are integrated as plug-ins and can be upgraded or extended independently without changing the core structure or design of the framework. Comparing with existing similar systems as well as statistical softwares in market, our framework is domain-independent with easily extensible interfaces and provides an entire suite of automatic experimental design, model execution, and analysis tools such that the domain experts can focus on the detailed studies instead of the formidable computing issues.

1.3 Paper Organization

The remainder of this paper is organized as follows. Section 2 introduces the architecture of framework, including the overall system design, the supporting infrastructures, and the core components. Section 3 discusses the technical aspects of the implementation. Section 4 gives an example application of using our framework to configure a simulation model and to conduct experiment design and analysis. Finally, we close the paper with a summary and direction of future work.

2 FRAMEWORK ARCHITECTURE

2.1 System Overview

In our framework, the working space is separated into a *general level* and a *native level*. Users perform the experimental design and analysis on the *general level* through the uniform interfaces we provide. Then the system transforms the simulation workflow and the configuration specification into a model-specific manner on the *native level* by using a script or program wrapped with the model. The data management and model execution issues inherent in experimental design for complex workflows are handled transparently from the user such that they can focus on the design itself.

The proposed system has several major components. First, a uniform specification mechanism in XML format defines the data class, data type, workflow specification, configuration format, and experimental design, etc. There are several advantages of using XML as specification format - it is simple, extensible, easy to parse, human and machine readable - as a matter of fact, XML format is widely adopted by similar systems like SAFE (Hallagan 2011) and JAMES II (Himmelspach, Ewald, and Uhrmacher 2008). The uniform specifications provide the user with a standard access point to different simulation models integrated into this framework, which is the foundation of the design. These XML specifications are stored in a central registry and all data and model execution requests will be sent to the central registry rather than the physical data storages or computational back-ends. The second component is the experimental design module. The user identifies the parameters (*factors*) with which to experiment and assigns corresponding values (*levels*). The combination of factors and levels forms the experimental design space. If an experimental design type is selected, the DOE module will sample the DOE space and generate experiment sites (*cells*). Third, a translation mechanism converting the workflow and configuration files in uniform format to the executable scripts or programs. This is taken care of by *model translator*, which is a model-specific script. Finally, an automated UQ/SA module performs systematic analysis on the experimental results. Such UQ/SA analysis provide insights on the adaptive experimental design and model validation/calibration. Note that the user performs the experimental design and analysis on the general level and the details of model execution and other native features are hidden from them.

2.2 Supporting Computational Infrastructures

Figure 2 gives a broad picture of how the entire procedure runs. The core component of our framework focuses on automating the experimental design and assisting UQ/SA studies. Before we proceed, it is necessary to introduce the supporting infrastructures.

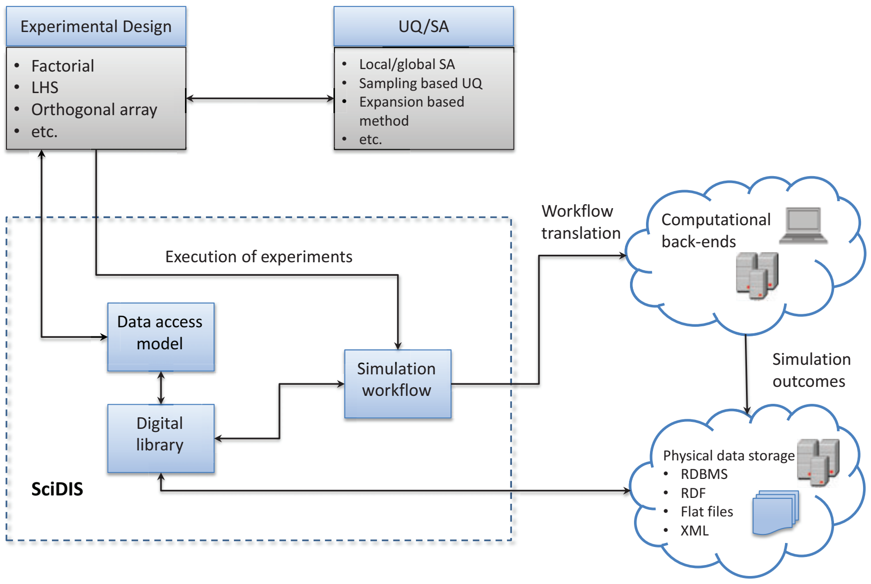

Figure 2: System overview.

First, SciDIS (Bisset, Mortveit, and NDSSL 2015) is a formally defined scientific digital information system that, in domain independent manner, handles complex workflow and their data in a completely recordable manner. It incorporates a digital library (DL) called SciDIL systematically managing the data sets, and a data access model handling the data requests and providing coherent access to them. The uniform simulation workflow specification is also stored in SciDIS as data class, such that the experimental design module will interact with the simulation model and process data in a generic manner. SciDIS provides the essential elements to handle a multitude of complex data formats in our system and serves as the aforementioned central registry.

Second, platforms such as Simfrastructure (Bisset et al. 2013) support a general environment for high performance computing (HPC). Model execution is absolutely model-specific, thus, this component is working basically on the native level. The scalability of the experiment generally depends on the scalability of the simulation model. Therefore, the user needs to decide how would the task be grained. More specifically, One could run the experiment instances one by one in sequential or run them in parallel. Moreover, a single simulation run can be distributed over a collection of machines whereas it can also be executed on a single machine coarsely.

Third, a physical data storage portion handles the actual data. It will deal with arbitrary combinations of the hardware and the storage methods (flat files, DBMS, RDF, etc.) and will interact with the digital library in the registry via generic APIs.

It is worth noting that, although the supporting environments are necessary, we do not assume any form of them but provide adapters connecting with particular infrastructure, so as to make our experimental design framework standalone and generic.

2.3 Uniform Specification Mechanism

Computer simulation model often consists of a collection of executables such as programs or scripts running on a wide variety of hardware with each transforming a collection of input data sets into output data sets as specified by one or more configurations. We have developed a precise abstraction in SciDIS for formalizing workflows with a translation mechanism mapping formal workflows into one or more workflow languages. The formal workflow description is a standard for a data class to specify general workflows along with data classes for workflow configuration files. All these data classes are associated with references to the central registry, such that all information are recorded and all experiments are repeatable. To formally describe a workflow, following necessary information are required: input data classes, output data classes, a configuration specification, and a workflow composition specification describing the dependency relations among the sub-models.

Configuration files contains essential information for model execution, such as the references of input and output data paths, the running environment, and the hardware resource requests, etc. Another important use of them is to expose possible factors for the design of the experiment. It is worth noting that configurations are often model-specific, and those who need to run the simulation models are usually expected to have skills such as developing and customizing their own scripts. This procedure often prevents novice users and application domain experts without computing backgrounds to run the simulations appropriately. In the system discussed in this paper, we provide a uniform format for both the workflow and configuration specification that only exposes information for model parameters and experimental design. The user will only work on the general level and other procedures such as translation, model execution, data management are hidden from them, see Figure 3. This makes the framework easy to use and flexible considering the challenges that many domain-specific systems face.

2.4 Experimental Design Module

Santner, Williams, and Notz (2003) point out that there are significant differences between traditional field experiments and computer experiments in the sense that the latter is deterministic, large-scale(with much more factors and levels) and resource-demanding. As a result, modern computer experiments adopt spacing-filling design techniques such as Latin hypercube sampling, orthogonal arrays and other mixed design methods. We incorporate a list of such computer experimental design algorithms as well as

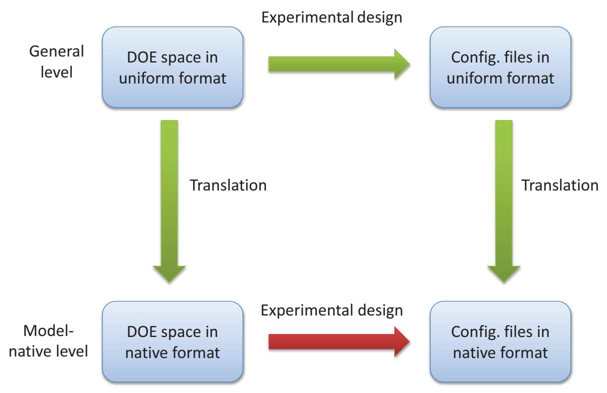
Experimental design
General level
DOE space in uniform format
Config. files in uniform format
Translation
Translation
Experimental design
Model-native level
DOE space in native format
Config. files in native format

Figure 3: Commute diagram of uniform specification format and native specification format.

many popular classical experimental design methods such as factorial design, central composite design, Box-Behnken Design and so on.

With the experimental design space specified, the user only needs to select the sampling algorithm and the experimental design module will produce the cells (sample points) automatically. The selected experimental cells are actually a collection of configuration files in the uniform format that will be translated into the model-specific format by the model translator in the pre-processing stage. To assist the UQ/SA study, we associate each experimental design instance and run with an ID that is recorded in the central registry facilitating data tracking and retrieval, such that one can easily obtain the map from the inputs to the outcomes.

We also explored more advanced adaptive design methods. The key idea of adaptive design is to sequentially choose the optimal points into the design such that the information retrieved from the design can be maximal. This usually involves a statistical surrogate model such as Gaussian process helping to predict the next optimal sample point(s) (Williams, Santner, and Notz 2000). The data access model makes it possible for adaptive design to inspect data in a uniform way. As a matter of fact, uncertainty quantification can also be enabled within the adaptive design process in a way that one can select the most uncertain points as the candidates.

2.5 Automated UQ/SA

UQ and SA are overlapped in many situations since they have common issues to address: how to represent the uncertainties and what are the effects of the uncertainties? Generally speaking, there are two types of uncertainties: aleatory uncertainty, also called random uncertainty, which is irreducible, and epistemic uncertainty which is the result of lack of knowledge (Roy and Oberkampf 2011). In our framework aleatory uncertainty is represented by probability distributions that can be specified through APIs we provide, and epistemic uncertainty is implemented by intervals (the range of a particular parameter).

We incorporate three types of UQ/SA methods: (a) analytic methods, (b) sampling-based methods, and (c) graphical methods. Analytic methods are often used in local sensitivity analysis, and expansion-base UQ (Crestaux, Le Maıtre, and Martinez 2009, Xiu and Karniadakis 2002). Sampling methods involve constructing statistical models, and perform the analysis by using methods such as analysis of variance (ANOVA), regression modeling, machine learning, Bayesian calibration and so on. Graphical sensitivity analysis provides the analysis in the form of graphs, charts, or surfaces which can assist screening design in the early stages and can also provide deliverable results. These various analysis can be specified in the input file of the framework by the users according to their particular studies and interests.

As described in the previous sections, this framework provides necessary components, including a formally specified workflow, automated experimental design, data access and management models, to support the UQ/SA kind studies for a wide variety of computer simulation models. The UQ/SA studies can in turn help to calibrate and validate the simulation models.

3 IMPLEMENTATION

3.1 Uniform Configuration format

The uniform configuration specification is formulated by using XSD/XML with one XSD file describing the configuration structure and the XML files being actual configuration files (uniform format that complying with the common XSD). Basically, there are three types of factors for experimental design: scalar-valued parameters such as transmissibility in the disease transmission model, a group of parameters that will vary at the the same time such as a sub-configuration file containing subordinated configuration keys, and input data classes. If we assume a tree architecture of a configuration file (Actually, the XML format inherently has a tree structure), the first two types of factors can be automatically exposed with scalar-valued parameters corresponding to the leaves and group parameters represented by the tree branches. For the input data classes, they can also be treated as leaves with levels being digit object labels stored in the central registry. Figure 4 gives a demonstration of the tree structure. It can be observed that branch B and C have the same sub-parameter D. To avoid ambiguity, it suffices to give the full path when accessing a parameter with the root node being omitted from the path.

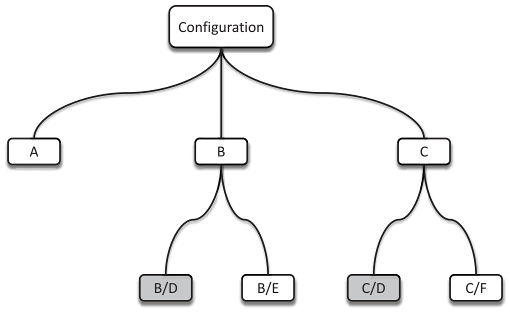

Figure 4: Tree structure of a configuration specification.

Recall that each simulation model has its own configuration style and format, however, they basically consist of a collection of key-value pairs. Some of the parameters are static, i.e., they stay unchanged during the whole simulation process, such as the version number, input and output path, etc. Other parameters need to be configured, among which, some are not appropriate to be the statistical factors whereas other parameters are those the simulation practitioner actually would like to experiment with. Based on this fact, we do not put too many constrains on the uniform configuration specification since it is supposed to accommodate a broad range of simulation models, but we have a separation of *exposed* and *non-exposed* parameters by marking the them as XML attributes such that the user can only focus on the parameters that are appropriate to be the statistical factors. As a concrete example, Figure 5 gives a XML configuration file for EpiFast - an epidemic propagation simulation model (Bisset et al. 2009) and Section 4 will discuss more details about it. In this example, “InterventionFile” is a sub-configuration file that specifies intervention schemes such as vaccination, school closure, and so on. It is represented as a tree branch with sub-parameters grouped underneath it as is shown in Figure 5.

```
<configSpec version="1.0">
  <registryInfo>
    <configurationName>EpiFastConfig</configurationName>
    <methodName>EpiFast</methodName>
    <description>Configuration specification for EpiFast</description>
    <baseConfigFile>EpiFastConfig:0001</baseConfigFile>
  </registryInfo>
  <configuration>
    <parameters>
      <parameter key="ConfigVersion" type="integer" exposed="false" optional="false">
        <value>2009</value>
      </parameter>
      <parameter key="OutputFile" type="string" exposed="false" optional="false">
        <value>/Path.To./EF06</value>
      </parameter>
      ...
      <parameter key="InterventionFile" type="string" exposed="false" optional="false">
        <value>/Path.To./Intervention</value>
        <parameter key="InterventionFile/InterventionVersion" type="integer" exposed="false" optional="false">
          <value>2010</value>
        </parameter>
        <parameter key="InterventionFile/Intervion1" type="string" exposed="false" optional="false">
          <value>I1</value>
          <parameter key="InterventionFile/Intervion1/SubpopulationId" type="integer" exposed="false" optional="false">
            <value>subpopulation:4710_5709</value>
          </parameter>
          <parameter key="InterventionFile/Intervion1/ActionEfficacy" type="float" exposed="true" optional="false">
            <value>0.5</value>
          </parameter>
          <parameter key="InterventionFile/Intervion1/ConditionCompliance" type="float" exposed="true" optional="false">
            <value>0.75</value>
          </parameter>
          <parameter key="InterventionFile/Intervion1/Stockpile" type="integer" exposed="true" optional="false">
            <value>2000</value>
          </parameter>
        </parameter>
      </parameter>
      ...
    </parameters>
  </configuration>
</configSpec>
```

Figure 5: Excerpt of an XML configuration specification of EpiFast.

3.2 Model Translator

One important role the model translator plays is to connect the general application level with the native model execution level. In particular, it handles the translation of the model description and configuration specifications in uniform format to and from machine readable model-native format. Since the uniform specifications are defined by XSD/XML, the translator is implemented by certain XML file parsers (such as lxml in Python, Xerces in C++). It also prepares the execution environment for the simulation model. It uses the run data files specified in the execution request along with command line parameters to create working directories, input files including configuration files, execute the model, deal with post-processing such as validate and summarize the results, create log file and tracking IDs registered in the central registry, and so on. We expect that we develop fairly generic translator that can be used with broad classes of models.

3.3 Sampling and UQ/SA Tools

Statistical design and analysis tools are excluded from the core component of this framework because they can be very problem specific. Based on particular application requirement, they can be integrated into our system as plug-ins. We have already implemented basic statistical design modules such as factorial design and Latin hypercube sampling; more statistical design algorithms will be included. The plug-ins are easy to add, remove, upgrade, and maintain through the APIs we provide, which increases the flexibility and extensibility of our framework.

4 Usage Example

In order to illustrate the concepts, we give a simulation example of studying the effects of public health strategies by using our framework. The focus in this section is on the usage of the framework and it will be extended in the future work.

4.1 Scenario

The 2014 Ebola outbreak in Western Africa has been identified the largest one in history and brought significant threat to human lives and society. Unfortunately, there is still no known cure, thus, combating it is challenging. Computational scientists have devoted a lot of efforts via computer simulations in modeling the disease outbreaks, forecasting the epidemic propagation, and analyzing medical facilities placement, see (Rivers 2014, Rivers et al. 2014).

In this example, we use simulation parameterized to produce the Ebola outbreaks similar to the 2014 Western Africa situation in Liberia. Then, ring vaccinations with different levels of compliance (the probability that an individual takes the vaccine), efficacy (the rate that the vaccine reduces the transmissibility of an infected individual) and stockpile of the vaccines are applied. The goal is to explore how the vaccination schemes affect the infectious size of the population.

4.2 Simulation Model Settings

The simulation is implemented by using EpiFast (Bisset et al. 2009). EpiFast is a simulation model of stochastic disease propagation in a large scale contact network. It has been used in real studies with rather sophisticated settings to evaluate various dynamic interventions like vaccination and to provide decision support for public health policy makers. For a typical experiment run, one have to perform following procedures:

1. Provide customized configuration file for EpiFast.
2. Compose a wrapper script to run the simulation code.
3. Submit the simulation job to the computational clusters by providing the qsub script.
4. Collect the raw simulation outcomes by interacting with the data storage systems.
5. Write a script to extract information of interest such as the infectious size, and conduct analysis by using certain statistical softwares such as Minitab and JMP.

One can see that each of the steps requires extensive expertises of programming, computation and the particular features of EpiFast.

4.3 Experimental Procedure and Results

Traditionally, experiments with EpiFast are set up manually. For experiment practitioners and domain experts without specialized knowledge of programming or computing, it is challenging to configure and execute a complex simulation model like EpiFast. In addition, for experiments with large number of runs, it is very inefficient and tedious to perform above-mentioned procedures by hand. Benefiting from the setting of our framework, it is much easier for those who only care about the analysis and study instead of the computational issues. They will work on the standardized uniform level to configure the simulation model, design the experiment, and conduct UQ/SA studies. The framework will handle the data management, model execution, output collection and inspection and so on.

For this experiment, there are three parameters: compliance rate of using vaccine, vaccine efficacy, and vaccine stockpile. If the user specifies the values of the parameters and select the type of experimental design, the system will generate the experimental sites with corresponding configuration files, run the model, and produce the simulation output in a recordable manner. As an example, a user specifies the levels for each factor as shown in Table 1 and plan to run a full factorial design.

Table 1: Selected levels for vaccine efficacy, compliance, and stockpile.

Factor	Levels
Efficacy	0.5,0.8
Compliance	0.25,0.5,0.75
Stockpile	1000,2000,5000,10000

The system will generate a collection of configuration files corresponding to the cells in the uniform format, and the model translator will translate each uniform configuration files into the model-native format as well as submit the execution request to the computational back-ends. The translation, model invocation and execution procedures are handled on a low level that is hidden from the user. The user can access the simulation results through the data access model of the central registry and perform analysis. Table 2 is the parameter effect test result for this experiment. It shows that efficacy and stockpile have a significant effect on the epidemic size of the outbreak since the p-value is less than 0.0001 whereas compliance does not have much impact on the response if we set the significance level $\alpha = 0.05$.

Table 2: Parameter effect test for full factorial design.

Factor	F ratio	Prob > F
Efficacy	33.5588	<0.0001
Compliance	3.6031	0.0722
Stockpile	37.5982	<0.0001

Assume the user plans to design the experiment in a finer way and samples efficacy, compliance from 0 to 1, and stockpile from 0 to 10000 uniformly using smaller intervals, say 10 levels for each factor. In this case, full factorial design will be too costly since there are total 1000 combinations. Then, a sparse design such as Latin hypercube sampling will help to reduce the number of runs significantly. The user only need to specify the design type as LHS and sample size, our framework will produce the experimental cells automatically.

This example demonstrates that the user will only focus on the experimental design and analysis of the simulation outcomes instead of handle the data management, model execution and other model-specific issues by using our framework. It will allow domain experts who are not necessarily computing experts to work more easily with complex simulation models like EpiFast.

5 DISCUSSIONS AND FUTURE WORK

In this paper, we propose a generic framework for conducting experimental design and UQ/SA for a broad range of simulation models. This framework is general in the sense that it is domain-independent and no particular computational infrastructure is assumed. By using it, the user does not need to handle the formidable data management, model configuration, invocation and execution as well as other mode-specific issues but only focus on the experimental design and analysis.

This work is at the preliminary stage and is being extended in many ways. First, currently the system only provides command-line type interfaces and it will be packaged up with a web-based GUI to improve the usability. Second, more analysis elements such as visualization tools will be implemented as plug-ins. Last but not least, we will explore designs to support adaptive design and advanced UQ/SA studies.

ACKNOWLEDGMENTS

We thank our external collaborators and members of the Network Dynamics and Simulation Science Laboratory (NDSSL) for their suggestions and comments. This work has been partially supported by DTRA Grant HDTRA1-11-1-0016 and DTRA CNIMS Contract HDTRA1-11-D-0016-0001. The authors would like to thank Dr. Bryan Lewis, Dr. Abhijin Adiga, Dr. Xinwei Deng, Dr. David Higdon and Mr. Md Maksudul Alam for many helpful discussions.

REFERENCES

Adams, B. M., W. Bohnhoff, K. Dalbey, J. Eddy, M. Eldred, D. Gay, K. Haskell, P. D. Hough, and L. Swiler. 2009. "DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 5.4 users manual". *Sandia National Laboratories, Tech. Rep. SAND2010-2183*.

Bisset, Keith and Mortveit, Henning S. and NDSSL 2015. "SciDIS and SciDIL – System and Digital Library Design". NDSSL Technical Report no. 15-020.

Bisset, K. R., J. Chen, X. Feng, V. Kumar, and M. V. Marathe. 2009. "EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems". In *Proceedings of the 23rd international conference on Supercomputing*, 430–439. ACM.

Bisset, K. R., S. Deodhar, H. Makkapati, M. V. Marathe, P. Stretz, and C. L. Barrett. 2013. "Simfrastructure: A Flexible and Adaptable Middleware Platform for Modeling and Analysis of Socially Coupled Systems". In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 506–513. IEEE.

Committee on Mathematical Foundations of Verification, Validation and Uncertainty Quantification, Board on Mathematical Sciences and Their Applications, and Division on Engineering and Physical Sciences. 2012. *Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification*. Washington, D.C.: The National Academies Press.

Crestaux, T., O. Le Maıtre, and J.-M. Martinez. 2009. "Polynomial chaos expansion for sensitivity analysis". *Reliability Engineering & System Safety* 94 (7): 1161–1172.

Ewald, R., J. Himmelspach, M. Jeschke, S. Leye, and A. M. Uhrmacher. 2010. "Flexible experimentation in the modeling and simulation framework JAMES IIimplications for computational systems biology". *Briefings in bioinformatics* 11 (3): 290–300.

Hallagan, A. W. 2011. *The design of XML-based model and experiment description languages for network simulation*. Ph. D. thesis, Bucknell University.

Hedayat, A. S., N. J. A. Sloane, and J. Stufken. 1999. *Orthogonal arrays: theory and applications*. Springer.

Himmelspach, J., R. Ewald, and A. M. Uhrmacher. 2008. "A flexible and scalable experimentation layer". In *proceedings of the 40th conference on winter simulation*, 827–835. Winter Simulation Conference.

Johnson, M. E., L. M. Moore, and D. Ylvisaker. 1990. "Minimax and maximin distance designs". *Journal of statistical planning and inference* 26 (2): 131–148.

McKay, M. D., R. J. Beckman, and W. J. Conover. 1979. "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code". *Technometrics* 21 (2): 239–245.

Perrone, L. F., C. S. Main, and B. C. Ward. 2012. "SAFE: simulation automation framework for experiments". In *Proceedings of the Winter Simulation Conference*, 249. Winter Simulation Conference.

Rivers, C. 2014. "Ebola: models do more than forecast". *Nature* 515 (7528): 492–492.

Rivers, C. M., E. T. Lofgren, M. Marathe, S. Eubank, and B. L. Lewis. 2014. “Modeling the impact of interventions on an epidemic of Ebola in Sierra Leone and Liberia”. *arXiv preprint arXiv:1409.4607*.

Roy, C. J., and W. L. Oberkampf. 2011. “A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing”. *Computer Methods in Applied Mechanics and Engineering* 200 (25): 2131–2144.

Santner, T. J., B. J. Williams, and W. Notz. 2003. *The design and analysis of computer experiments*. Springer.

Williams, B. J., T. J. Santner, and W. I. Notz. 2000. “Sequential design of computer experiments to minimize integrated response functions”. *Statistica Sinica* 10 (4): 1133–1152.

Xiu, D., and G. E. Karniadakis. 2002. “The Wiener–Askey polynomial chaos for stochastic differential equations”. *SIAM Journal on Scientific Computing* 24 (2): 619–644.

AUTHOR BIOGRAPHIES

Sichao Wu is a Ph.D. student in the Department of Computer Science at Virginia Tech. He is advised by Dr. Henning S. Mortveit and works on the theoretical foundations of agent-based simulation models. He earned B.S. and M.S. degrees in Computer Science from Chongqing University, China. His email address is sichao@vt.edu.

Henning S. Mortveit is an Associate Professor of Department of Mathematics and a Senior Research Associate in the Network Dynamics and Simulation Science Laboratory VBI at Virginia Tech. He received his doctorate in Mathematics from the Norwegian University for Science and Technology. His research involves structural analysis of dynamical processes over networks, large scale computer simulation models, and the modeling and design of such systems. His e-mail address is henning.mortveit@vt.edu.