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 MODEL-DRIVEN ENGINEERING APPROACH TO SIMULATION EXPERIMENT DESIGN AND EXECUTION

Alejandro Teran-Somohano
Alice E. Smith

Department of Industrial and Systems Engineering
Auburn University
Auburn, AL 36849, USA

Joseph Ledet
Levent Yilmaz

Department of Computer Science and Software Engineering
Auburn University
Auburn, AL 36849, USA

Halit Oğuztüzün

Department of Computer Engineering
Middle East Technical University
Ankara, Turkey

ABSTRACT

There is an increasing awareness of the added value that efficient design of experiments can provide to simulation studies. However, many practitioners lack the proper training required to design, execute and analyze a well-designed simulation experiment. In this study, we present the initial stages of a model-driven engineering based tool for managing simulation experiments. Underlying our approach are an experiment ontology and a feature model that capture the statistical design of experiments expert knowledge which users might be lacking. In its current state, the tool provides support for the design and execution of simple experiments. Using a web-based interface, the user is guided through an experiment design wizard that produces an experiment model that can be exported as an XML file containing the experiment's description. This XML can be used to synthesize scripts that can be run by a simulator and shared across different platforms.

1 INTRODUCTION

As the use of simulation models for scientific research, industry, and public policy has become more prevalent, there has been a parallel increase in the need for better experimental practices and cross-validation of experimental results. Scientific knowledge is required to be both reproducible and replicable, and the absence of appropriate means for this has generated a credibility gap in the field of modeling and simulation (Yilmaz and Ören 2013).

Many researchers have suggested that one effective way of supporting reproducibility and replicability in modeling and simulation (M&S) is by making use of statistical design of experiments (DoE). DoE has provided scientists with a means of reproducing and replicating physical experiments for many years. By making some adaptations to its methodology, we can use it for the same purpose in modeling and simulation (Sanchez and Wan 2012; Lorscheid, Heine, and Meyer 2012; Kelton 2000; Kleijnen 2005). In our previous work, we have taken a similar position (Ledet et al. 2014) and have proposed using, in addition to DoE, a Model-Driven Engineering (MDE) framework for managing simulation experiments throughout their entire life-cycle (Teran-Somohano et al. 2014). Our proposed approach was built on three tenents: (1) model-driven engineering, (2) agent-assisted workflow systems,

and, (3) design of reproducible experiments. In this paper, we discuss some advances we achieved in implementing such a system. The majority of the work found in the literature focuses on the proper techniques required for designing simulation experiments (Kleijnen 2005; Kelton 2000; Lorscheid, Heine, and Meyer 2012; Sanchez and Wan 2012). Though there are emerging tools for managing experiments (Ioannidis et al. 1997; Perrone, Main, and Ward 2012; Leye and Uhrmacher 2012), they do not encompass the entire experimental life-cycle, which includes adaptation of experiments as learning takes place.

A recent effort in supporting users in managing simulation experiments is that represented by SESSL (Ewald and Uhrmacher 2014), an internal domain-specific language for simulation experiments. There are, however, important differences between our work and SESSL. Whereas SESSL is mainly concerned with the specification and execution of the experiment using an imperative general-purpose programming language, our system accommodates the incremental iterative nature of experiments and uses declarative models to drive experiment execution. Furthermore, SESSL does not include DoE principles as part of its grammar. Our focus is on guiding the user in designing experiments that conform to the principles of DoE. Regardless, it seems possible to integrate SESSL with our approach, with our system producing experiment descriptions that conform to DoE principles, and SESSL synthesizing experiment specifications from those descriptions, and executing them efficiently.

2 BACKGROUND

Of our three tenents, MDE, agent-based workflow systems, and statistical design of experiments (DoE we have developed the first and the last of these three. In this section, we present some background information on these two topics. For greater detail, please see (Teran-Somohano et al. 2014).

2.1 Statistical Design of Experiments

M&S has much to gain from building upon the statistical theory of design of experiments. As (Kleijnen et al. 2005) affirm, simulation practitioners have put too much emphasis on building their models and not on analyzing them. Most simulation models are analyzed by changing one factor at a time, neglecting the potential impact of interactions among factors. Using DoE methods, practitioners can explore many alternative configurations of the model at an efficient computational cost. As a result, they can gain greater insight and knowledge about the system they have modeled (Sanchez and Wan 2012). Figure 1 presents a high-level experiment life-cycle that follows the DoE methodology. More detailed discussions on DoE can be found in classic text books such as (Montgomery 1997) or in (Kleijnen et al. 2005) as it applies to simulation experiments directly. Here, we will cover its basics very briefly.

An *experiment* is a systematic, iterative process aimed at answering a set of questions about the system being studied. The questions define the experiment's *objectives*. Once the objectives are defined, the model's *variables* or parameters are categorized as dependent, independent, nuisance, or control variables. Each kind of variable is mapped to either a *response* (if dependent) or a *factor* (otherwise). Responses correspond to output values from the simulation model, whereas factors correspond to input parameters. The term factor is DoE jargon for model parameter. For each factor, we also have to define its *levels*, that is, the range or series of values that it can take. Based on these decisions, we can then select an *experiment design*, which describes a method for generating *factor level combinations* that can help characterize the response surface. See (Sanchez and Wan 2012) for a useful overview of designs for simulation experiments. DoE-based designs are sampling strategies that have been developed to be efficient in collecting data about a system's response. They have also been designed to offer support for statistical techniques that can quantify the effects of factors on responses, the size and direction of interactions among factors, etc. The specific design to be used depends on many different criteria, including the experiment's objective, the number of experiment factors and their corresponding levels, the availability of resources for experimentation, the desired precision. Our approach seeks to supply the connection between these criteria and specific experiment designs, making the experiment design process

as transparent as possible. These connections are captured in both the ontology and the feature model, and are used to guide the user throughout the design process.

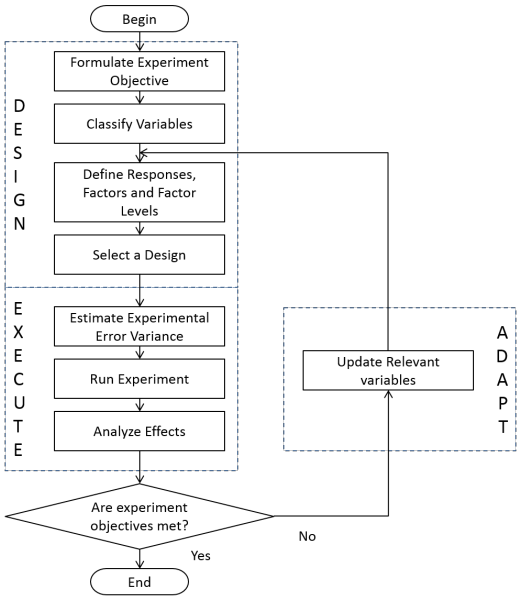

Figure 1: Experiment life-cycle.

Once a certain experimental design has been selected we can construct a *design matrix*, representing all factor level combinations to be explored. Each row in the matrix stands for a sampling point to be collected. We can then proceed to run the experiment, retrieve its outcome and commence analyses. These can include graphical analyses (histograms, scatter plots, etc.), statistical analyses (ANOVA, regression) or a combination of both. If the experiment's objectives are not met, or if new insights or questions emerge from the experiment's outcome, one can modify the current experiment (by using a different set of input parameters, for example) and generate a new design. This adaptation of the experiment is not covered in this paper, but is part of our ongoing work. This iterative process of design, execution, and adaptation of simulation experiments is what we have termed the experiment life-cycle.

2.2 Model-Driven Engineering

As shown in Figure 2, the MDE methodology (Gaševic et al. 2006) provides a framework and strategy for moving from platform-specific to platform-independent domains and back. In our approach, an ontology and a feature model are used to capture the expert DoE knowledge. Both models conform to the principles of DoE. The experiment model can be exported as an XML description that conforms to the ontology (in its static, structural elements) and to the feature model (in its variables components). This platform independent representation can be exported and parsed to generate executable scripts that conform to the specific language used by simulator at hand.

The use of MDE allows the de-coupling of system knowledge from system execution. Changes to the models do not require changes to the code base, meaning we can update the expert knowledge of the system without modifying code. It also permits the integration of additional software components that can process and exploit the knowledge base, as will be discussed in the conclusions. Furthermore, the MDE approach allows for greater transparency, in that the DoE process and principles are made explicit. By interacting with the models, the impact of the user's decisions is immediately manifested in the range of decisions that can be made as the design process advances. The models ensure that the user does not attempt to design and execute experiments that do not conform to the principles of DoE.

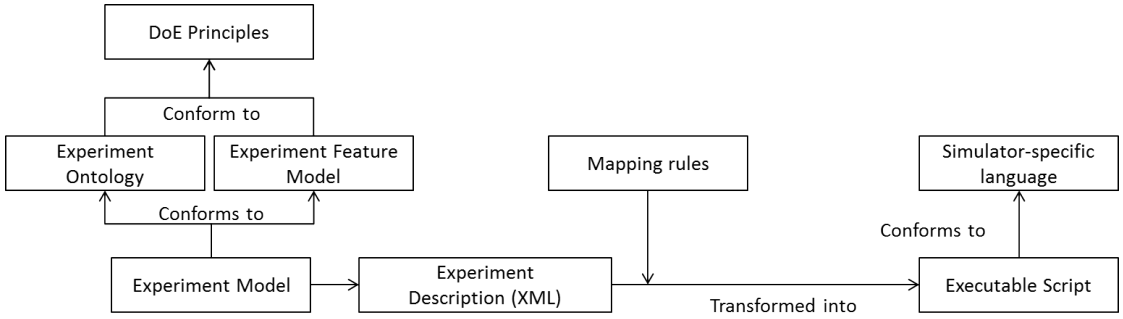

Figure 2: MDE for experiment management.

3 MODEL DESCRIPTION

This section briefly describes the two main models that form the basis of our expert knowledge, the experiment ontology and the experiment feature model. The experiment ontology is primarily concerned with the static or structural components of an experiment. It describes the relevant terms and the relationships among those terms. The feature model, on the other hand, addresses the variable perspective of an experiment. It can be understood as a view on the experiment ontology that captures the range of variation that can exist between different experiments.

3.1 Simulation Experiment Ontology

In Section 2.1, we described the experiment life-cycle and highlighted certain important DoE terms. These terms are the building blocks of the ontology and form the basic vocabulary of DoE. The ontology captures not only these terms as abstract classes or concepts, but also the relationships that exist among different terms, together with the cardinalities governing these relationships. These relationships are modeled through object properties. For instance, the two terms *experiment* and *iteration* are linked together by the object property *has*, capturing the DoE principle that "an experiment has one or more iterations." Figure 3 shows a reduced version of our DoE ontology that is used for the prototype discussed in this paper. It includes only the basic concepts of an experiment. A more complete version of the ontology is found in (Teran-Somohano et al. 2014). Each of these abstract classes or concepts can be instantiated by an individual. An individual is the concrete realization of a concept of the ontology, and is often bound to a set of data values through datatype properties. For example, the class *Factor_Level* contains two datatype properties: *Level_number* and *value*. An individual, say Factor_Level001, will then have these two properties with specific values, e.g., 2 and 0.25.

In this work, the ontology is used in a manner akin to an XML schema. However, there are significant reasons why we chose an ontology instead of a schema. While an XML schema guarantees the integrity of an XML document, it does so statically, by checking if an XML document conforms to a predefined structure. An ontology does that and more. With the ontology, a data structure (i.e., an experiment model) is dynamically created that contains the relevant concepts required by the domain and conforms to the predefined structure of an experiment.

3.2 Simulation Experiment Feature Model

The feature model governs those components that vary from experiment to experiment. While all experiments have factors and responses, different experiments might have different number of factors and responses, or varying number of factor levels. These variable components are modeled as features. The number of factors and factor levels has an impact on the type of designs that can be used for the experiment. That is, the selection of certain features can limit the number of other available features. These interactions among features are modeled as constraints.

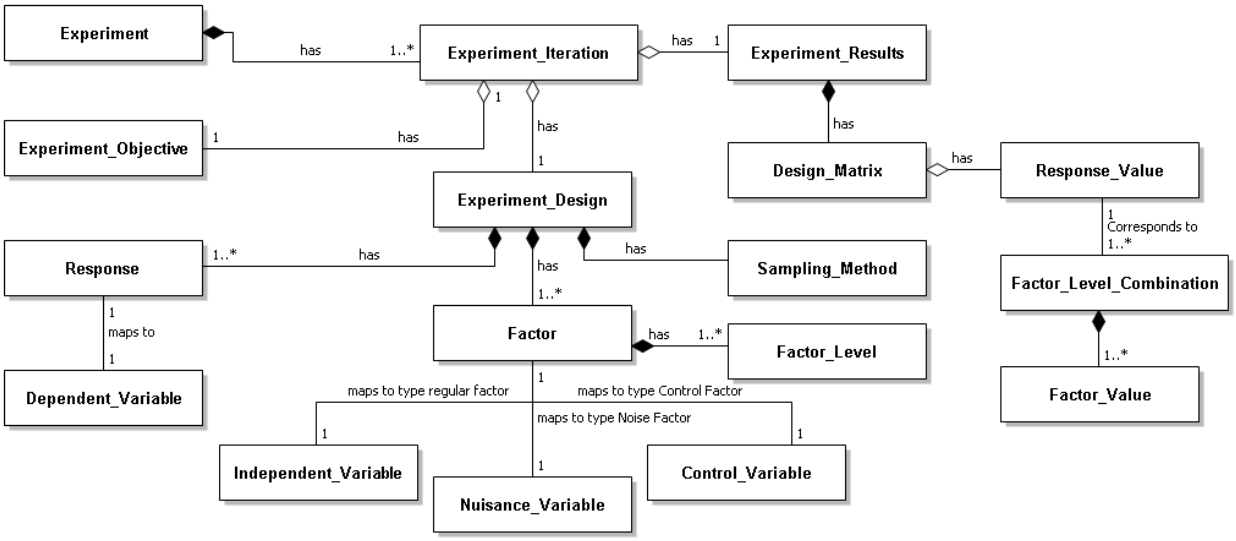

Figure 3: Selected concepts from the DoE ontology.

Figure 4 presents the feature model used by our prototype. The feature model is stored as a hierarchy of features, or a feature tree. A configuration is a combination of selected feature values. A configuration is valid if it does not violate any of the model constraints. Abstract features (shown in a lighter shade) are features that cannot be selected, but provide conceptual structure to the feature tree. Concrete features can be selected and constitute the configuration. Our feature model includes constraints that relate the number of factors, factor levels, computational cost, and the number of model replications to certain candidate experimental designs. For example, one of the constraints states that if there are two or three factor levels with between two to four factors and low or medium computational cost, a full factorial design is a good option. The development of the rules and relationships that form these constraints is part of our ongoing work.

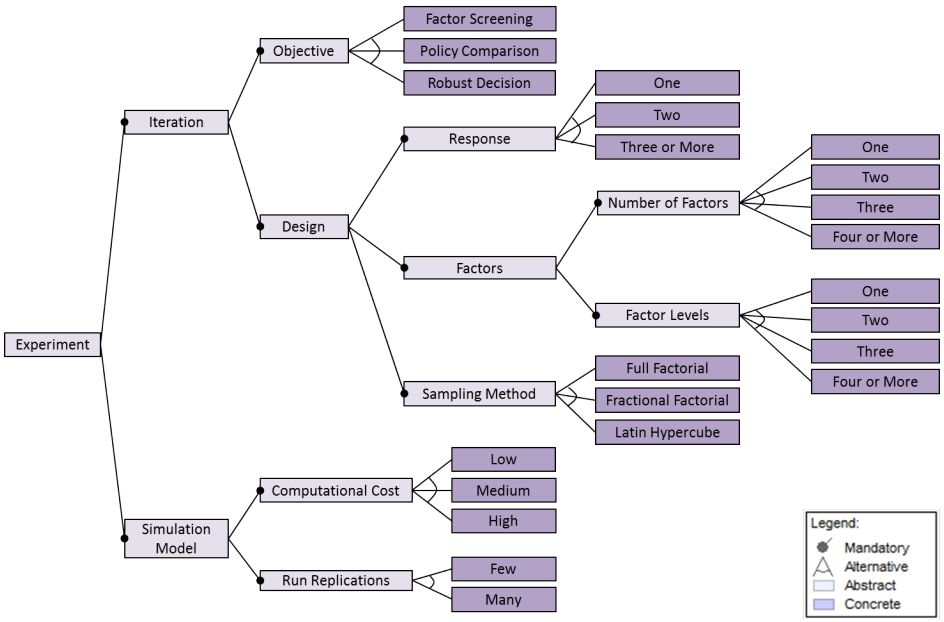

Figure 4: Experiment feature model.

3.3 Linking the Ontology and the Feature Model

The ontology and the feature model are linked at two different levels, one abstract and one concrete. At the abstract level, ontology concepts map to abstract features. Abstract features cannot be selected, they only provide conceptual structure to the feature tree. This allows querying of the feature tree. For instance, the ontology concept "SamplingMethod" corresponds directly to an abstract feature of the same name. At the concrete level, ontology individuals' data values are mapped to concrete features. So, under the abstract feature "SamplingMethod", we find three features "FullFactorial", "FractionalFactorial", and "LatinHyperCube." Whichever of these features that the user selects is used as the actual value of the ontology individual "SamplingMethod001."

4 SYSTEM DESIGN AND IMPLEMENTATION

The prototype was built as a web-application written in JSP and Java using OWL as the ontology language and FeatureIDE to implement the feature model. In this section, we describe the system's design, as well as selected aspects of its implementation.

4.1 System Architecture

Our system architecture is based on the mediator design pattern, which is used to encapsulate the interactions among different components, known as colleagues (Gamma et al. 1994). Given the variety of components that are required to interact with each other, the mediator pattern offers an effective and proven solution with flexibility and scalability. Figure 5 shows a diagram of the system architecture. The user interacts with the system through a web-based interface. This interface instantiates a mediator component that governs the interactions with the rest of the system. The models are accessed through manager components that make handling a transparent task for the mediator. In order to execute an experiment, the mediator interacts with a simulator interface that implements the mapping rules that allow the translation from the experiment description to an executable script.

4.2 Mediator

The mediator is the coordinator that connects all components together. Some of the functions it performs include: generating user interface screens, communicating with the ontology and the feature model managers, invoking the generation of the design matrix, and initiating experiment execution. Its future functionality will include: retrieving the output of the simulation runs, processing the output and invoking statistical packages for generating graphical and statistical analyses.

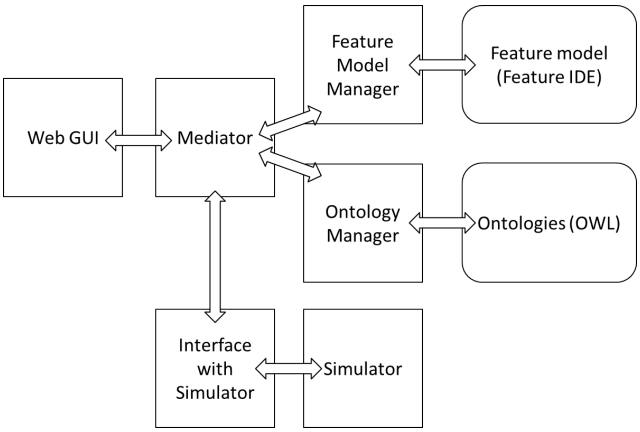

Figure 5: Prototype architecture.

4.3 Web-Based Interface

The web-based interface is automatically generated by the mediator based on the current stage of the design process. Knowing the information required at each stage, the mediator selects the controls that are presented to the user. This knowledge is contained in a GUI ontology. The GUI ontology describes the relationship between experiment components and the user interface controls. To generate screens automatically, the following information must be defined in the ontology: the specific control type (text box, drop-down box, etc.), the source of data for that control, the data types that will be accepted by the control, as well as any restrictions on the type of data to be captured.

In the current implementation, there are two types of controls available: text boxes and drop down boxes. Components can be configured in three ways: (1) by direct user input of a value, (2) by selecting from a dynamic list of values (the available values being determined by the user's previous choices), or (3) by selecting from a fixed list of values. These correspond to the three data sources available: (1) direct user input (by typing a value), (2) feature model, and (3) list of fixed values. The feature model data source allows leverage of the constraint enforcing functionality of FeatureIDE to prune the feature tree, so that the user can only select values that would result in a valid configuration, given previous selections.

The following are examples of these three data sources in operation: (1) The control used for specifying the number of factors in the experiment has direct user input as its data source. The user types in directly how many factors will be present. (2) The control used for selecting the experiment's sampling method has the feature model as its data source, because the list of available designs will change based on the user's previous selections. This functionality is included in the feature model. (3) The control used for classifying a variable uses a fixed list of values as its data source, because variables can be of only four types: dependent, independent, noise or control. This approach offers great flexibility in terms of a user interface. For example, if we wanted to change the control used for capturing the number of factors from a text box to a numeric input box, the ontology is updated with a numeric input box individual and linked to the number of factors entity. The interface should automatically be able to generate screens with the new control.

Consider an example. Suppose the current design phase is the selection of a sampling method. This implies that the user has already determined the experiment's objective, has classified the variables, and selected the factors to study, as well as their different levels. The mediator then invokes the ontology manager which, in turn, queries the ontology for the relevant presentation information. The ontology responds that the control to use is a drop-down box and that the data source is the feature model. The mediator then, requests that the feature model manager query the feature model for the list of features found under the abstract feature *SamplingMethod* that are still available. The feature model returns two: *FullFactorial* and *FractionalFactorial*. Given the user's previous choices, the *LatinHyperCube* feature is no longer available. Now that all of the required information has been collected by the mediator, it can generate the screen and wait for the user's input. In this case, the user selects one of the available options from the drop-down box.

4.4 Model Managers

Both the ontology manager and the feature model manager operate as interfaces between the mediator and the corresponding models. The ontology manager, in particular, allows the mediator to run queries on the ontology, but it also provides functionality for building XML files based on the ontology structure. This is essential for generating the experiment description. The feature model manager also allows the running of queries on the feature model, but, in addition, it creates and edits configurations, validates them, and enforces feature selection constraints.

4.4.1 Ontology Manager

The ontology manager interfaces with the two ontologies of the system: (1) the DoE ontology and (2) the GUI ontology by using the OWL API (Horridge and Bechhofer 2011). The way it generates the user interface was described in Section 4.3. In this section, how instantiates experiment models is described.

An experiment is modeled as a collection of individual components conforming to the DoE ontology specification. Individuals are related to each other through object property instances. At the top of Figure 6, there is a diagram of OWL classes from the DoE ontology. Each concept is related to the others by two object properties: *has* and *belongsTo*. The bottom portion of the figure shows an experiment model that instantiates the classes and properties of the upper portion. For example, the individual "ExperimentDesign001" *has* two individuals of the class Factor, namely, "Factor001" and "Factor002". The inverse property *belongsTo* exists between "Factor001", "Factor002" and "ExperimentDesign001."

The object properties are used to create a hierarchical structure in the experiment model that allows navigation from one individual to another, as well as determination whether more complex relationships between them exist. We can determine, for instance, that "Factor001" is a child of "ExperimentDesign001" (and that the latter is a parent of the former), or that "Factor001" and "Factor002" are siblings. Because of this, there is functionality in the ontology manager for adding an individual, given the new individual's parent, by automatically inserting it into its corresponding place in the experiment's structure. Individuals also have associated data. In Figure 7, examples of datatype properties are found in "FactorLevel001" which has two such properties: *value* and *type*. The ontology manager is capable of adding and modifying datatype properties in specific components of the experiment model.

At the end of the experiment design stage, our tool will have built an experiment that is consistent with the experiment ontology. We can use that experiment to generate an XML description directly. The ontology manager does this by exporting the hierarchy of individuals generated by OWL.

4.4.2 Feature Model Manager

FeatureIDE handles all the feature model functionality, so the feature model manager is only concerned with querying the feature tree and selecting features. Features can exist in three states: (1) selected, (2) unselected, and (3) undefined. A feature can be selected by either direct user selection or automatically through a constraint (as the result of the selection of another feature). Features can be unselected only through constraints. Any feature that is neither selected nor unselected is undefined. Undefined is the default state of any feature. One can query the model for the status of a given feature or one can retrieve an entire set of features based on their state. For instance, one can query for a list of all undefined features. These are the features that the user is free to select at that given point in time.

The feature model manager navigates the feature model tree through the abstract features. The name of a feature corresponds to that of a concept defined in the ontology. Therefore, once a concept from the ontology has been obtained, it can be used to retrieve its equivalent feature from the feature model, given the one to one correspondence between them. If what is required is a concrete feature, the feature model manager can access its parent abstract feature and then query it for all of its children. From this group, the feature model manager returns only those features that are set to undefined. Features that have been selected or unselected previously (either by the user directly or through a constraint) cannot be modified at this point. This ensures that the resulting configuration is valid.

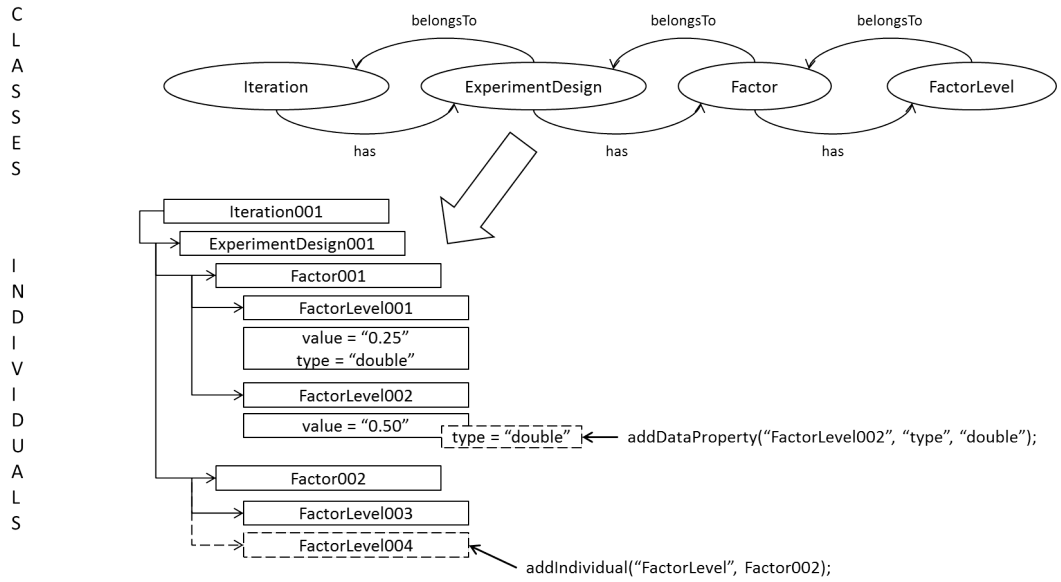

Figure 6: Experiment construction using the DoE ontology.

5 CASE STUDY USING THE SCHELLING MODEL

As a case study, a prototype of our tool is used to design and run several different experiments using the Schelling model provided as an example with RePast. In this section, we will briefly describe the Schelling model, and walk through the experiment design and execution using the proposed tool.

5.1 The Schelling Model

The Schelling model is a model used to study how patterns of racial segregation emerge in residential neighborhoods. The population of the neighborhood is divided into racial groups, with agents placed randomly at the initial stage. Agents are assumed to care about the color of their neighbors and can move to different locations if they are dissatisfied with the mixture of racial groups surrounding them (Schelling 1971). For our case study, the number of groups is set to three. We use one model output: the final average number of individuals belonging to the same group. We assume that the model has already been implemented, tested and validated in *RePast*, and that an XML description of its inputs and outputs exists.

5.2 Designing the Experiment

Suppose the user wishes to study the effects of changing two parameters: percentLikeNeighbors and maxDeathAge, that is, the user wishes to screen both factors to see whether they have a statistically significant impact on the response. Navigating through the web-based experiment design wizard, the user selects the objective "Factor Screening"; selects the two factors of interest, and decides to study them at two levels: 0.25 and 0.50 for the first level, 85 and 95 for the second level.

Based on this input, the system restricts the types of designs available to "Full Factorial." Had the user chosen a different combination of number of factors and factor levels, the system would have allowed alternative designs. For instance, with a high number of factors at two levels, it might have suggested instead a fractional factorial. Once the user has finished designing the experiment, the mediator invokes the experiment matrix designer agent to generate a design matrix based on the user's input. The mediator then takes the design matrix, and the user's input and inserts them into the experiment model. Using the description found in the model a *RePast* parameter sweep XML file is generated. RePast's batch run function is then called to run the model using the parameter sweep file.

5.3 Output

The end result of the prototype is an ontology-based experiment model, containing all of the information about the experiment, from its design through its results. This model as an XML that can be exported. This XML can be used to reproduce the experiment model for use with the same simulator or, in the future, replicating it with a transformed simulation model in a different simulator. The sample XML that corresponds to the case study Full Factorial with 2 factors and 2 factor levels is displayed in Figure 7.

6 CONCLUSIONS

This paper introduced a prototype of the design and execution component of a simulation experiment management system based on the principles of MDE and DoE. The objective of this work is to provide support for experiment reproducibility and replicability by providing DoE expert knowledge using an interactive interface that guides users through the experiment life-cycle.

```
<experiment>
  <iteration number="1">
    <objective value="FactorScreening" />
    <design name="Full Factorial">
      <response name="averageTypeLikeMe" type="double" />
      <factor id="1" name="percentLikeNeighbors" type="double">
        <factor_levels>
          <level id="1" value="0.25"></level>
          <level id="2" value="0.50"></level>
        </factor_levels>
      </factor>
      <factor id="2" name="maxDeathAge" type="int">
        <factor_levels>
          <level id="1" value="85"></level>
          <level id="2" value="95"></level>
        </factor_levels>
      </factor>
    </design>
    <results>
      <experiment_matrix>
        <factor_level_combination id="1">
          <response_value name="averageTypeLikeMe" value="" />
          <factor name="percentLikeNeighbors" type="double" value="0.25" />
          <factor name="maxDeathAge" type="int" value="85" />
        </factor_level_combination>
        <factor_level_combination id="2">
          <response_value name="averageTypeLikeMe" value="" />
          <factor name="percentLikeNeighbors" type="double" value="0.50" />
          <factor name="maxDeathAge" type="int" value="85" />
        </factor_level_combination>
        <factor_level_combination id="3">
          <response_value name="averageTypeLikeMe" value="" />
          <factor name="percentLikeNeighbors" value="0.25" />
          <factor name="maxDeathAge" value="95" />
        </factor_level_combination>
        <factor_level_combination id="4">
          <response_value name="averageTypeLikeMe" value="" />
          <factor name="percentLikeNeighbors" value="0.50" />
          <factor name="maxDeathAge" value="95" />
        </factor_level_combination>
      </experiment_matrix>
    </results>
  </iteration>
</experiment>
```

Figure 7: Experiment model exported as an XML

This work has been developed along two dimensions: one conceptual and one technical. The former dimension encompasses the theory of DoE, and has consisted mainly in the making explicit of those DoE principles that link various criteria to specific experimental designs. The current methods for choosing between candidate experimental designs rely mainly on generic rules of thumb, which are described in most of the literature. We make these rules explicit so that they can be translated into constraints for our models. Our future work will delve much more deeply along this dimension. Linking these criteria to

designs is the most important task and provides the bulk of the DoE expertise we wish our system to capture. So far we have linked number of factors and factor levels, as well as computational cost, to a few experiment designs. We have yet to do this for experiment objectives, and for many other designs that have been developed by statisticians.

The technical dimension covers the integration of different MDE techniques, such as ontology modeling, feature modeling, and model transformation and interconnection. The present work demonstrates that we have successfully connected the ontology with the feature model, resulting in an experiment model that can be transformed into an executable script. Despite that, there is much work left to do in further developing the models and meta-models. For the ontology, we still need to include the full range of terminology and relationships that exist in DoE. The feature model needs to be refined with a full set of constraints that can offer the user comprehensive guidance. We also need to develop the tool to encompass the entire experimental life-cycle. In its current stage, it only addresses the first two main stages: design and execution. A significant portion of our future work will be dedicated to the adaptation stage, and will include the use of intelligent agents for making decisions as to the best flow for the experimental process.

REFERENCES

Ewald, Roland, and Adelinde M. Uhrmacher. 2014. "SESSL: A Domain-Specific Language for Simulation Experiments." *ACM Trans. Model. Comput. Simul.* 24 (2): 11:1–11:25.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

Gaševic, Dragan, Dragan Djuric, and Vladan Devedžic. 2006. *Model Driven Architecture and Ontology Development*. Springer Science & Business Media.

Horridge, Matthew, and Sean Bechhofer. 2011. "The OWL API: A Java API for OWL Ontologies." *Semantic Web* 2 (1): 11–21.

Ioannidis, Yannis E, Miron Livny, Anastassia Ailamaki, Anand Narayanan, and Andrew Therber. 1997. "ZOO: A Desktop Experiment Management Environment." In *ACM SIGMOD Record*, 26:580–83. ACM.

Kelton, W David. 2000. "Design of Experiments: Experimental Design for Simulation." In *Proceedings of the 32nd Conference on Winter Simulation*, 32–38. Society for Computer Simulation International.

Kleijnen, Jack PC. 2005. "An Overview of the Design and Analysis of Simulation Experiments for Sensitivity Analysis." *European Journal of Operational Research* 164 (2): 287–300.

Kleijnen, Jack PC, Susan M Sanchez, Thomas W Lucas, and Thomas M Cioppa. 2005. "State-of-the-Art Review: A User's Guide to the Brave New World of Designing Simulation Experiments." *INFORMS Journal on Computing* 17 (3): 263–89.

Ledet, Joseph, Alejandro Teran-Somohano, Zachary Butcher, Levent Yilmaz, Alice E. Smith, Halit Oğuztüzün, Orçun Dayıbaş, and Bilge Kaan Görür. 2014. "Toward Model-Driven Engineering Principles and Practices for Model Replicability and Experiment Reproducibility." In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, 27:1–27:8. DEVS '14. San Diego, CA, USA: Society for Computer Simulation International.

Leye, Stefan, and Adelinde M Uhrmacher. 2012. "GUISE-a Tool for GUIding Simulation Experiments." In *Proceedings of the Winter Simulation Conference*, 305. Winter Simulation Conference.

Lorscheid, Iris, Bernd-Oliver Heine, and Matthias Meyer. 2012. "Opening the 'Black Box' of Simulations: Increased Transparency and Effective Communication through the Systematic Design of Experiments." *Computational and Mathematical Organization Theory* 18 (1): 22–62.

Montgomery, Douglas C, Douglas C Montgomery, and Douglas C Montgomery. 1997. *Design and Analysis of Experiments*. Vol. 7. Wiley New York.

Perrone, L Felipe, Christopher S Main, and Bryan C Ward. 2012. "Safe: Simulation Automation Framework for Experiments." In *Proceedings of the Winter Simulation Conference*, 249. Winter Simulation Conference.

Sanchez, S.M., and Hong Wan. 2012. "Work Smarter, Not Harder: A Tutorial on Designing and Conducting Simulation Experiments." In *Simulation Conference (WSC), Proceedings of the 2012 Winter*.

Schelling, Thomas C. 1971. "Dynamic Models of Segregation." *Journal of Mathematical Sociology* 1 (2): 143–86.

Teran-Somohano, Alejandro, Orçun Dayıbaş, Levent Yilmaz, and Alice Smith. 2014. "Toward a Model-Driven Engineering Framework for Reproducible Simulation Experiment Lifecycle Management." In *Proceedings of the 2014 Winter Simulation Conference*, 2726–37. IEEE Press.

Yilmaz, Levent, and Tuncer Ören. 2013. "Toward Replicability-Aware Modeling and Simulation: Changing the Conduct of M&S in the Information Age." In *Ontology, Epistemology, and Teleology for Modeling and Simulation*, edited by Andreas Tolk, 44:207–26. Intelligent Systems Reference Library. Springer Berlin Heidelberg.

AUTHOR BIOGRAPHIES

ALEJANDRO TERAN-SOMOHANO is a PhD candidate in Industrial and Systems Engineering at Auburn University. He received his B.S. degree in Computer Engineering at the Instituto Tecnológico Autónomo de México (ITAM), and his M.S. in Industrial Engineering from Auburn University. His email address is ateran@auburn.edu.

ALICE E. SMITH is the W. Allen and Martha Reed Professor of Industrial and Systems Engineering at Auburn University with a joint appointment in Computer Science and Software Engineering. She has authored papers with over 2,100 ISI Web of Science citations and has been a principal investigator on projects with funding totaling over $7 million. She is an area editor of *INFORMS Journal on Computing* and *Computers & Operations Research* and an associate editor of *IEEE Transactions on Evolutionary Computation* and *IEEE Transactions on Automation Science and Engineering*. Her email address is smithae@auburn.edu.

JOSEPH LEDET is a PhD candidate in Computer Science and Software Engineering at Auburn University. He received his B.S. degrees in Electrical Engineering and Computer Engineering from Louisiana State University and his M.S. in Computer Engineering from the University of Louisiana at Lafayette. He also has over eight years of experience as a software developer for private industry and government contractors. His email address is jwl0008@auburn.edu.

LEVENT YILMAZ is Professor of Computer Science and Software Engineering at Auburn University with a joint appointment in Industrial and Systems Engineering. He holds M.S. and Ph.D. degrees in Computer Science from Virginia Tech. He is the founding organizer and General Chair of the Agent-Directed Simulation Conference series and is currently Editor-in-Chief of *Simulation: Transactions of the SCS*. His email address is yilmaz@auburn.edu.

HALIT OĞUZTÜZÜN is an associate professor in the Department of Computer Engineering at Middle East Technical University (METU), Ankara, Turkey. He obtained his BSc and MSc degrees in Computer Engineering from METU in 1982 and 1984, and PhD in Computer Science from University of Iowa, Iowa City, IA, USA in 1991. His current research interests include model-driven engineering and distributed simulation. His email address is oguztuzn@ceng.metu.edu.tr.