Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

MODELS AS SELF-AWARE COGNITIVE AGENTS AND ADAPTIVE MEDIATORS FOR MODEL-DRIVEN SCIENCE

Levent Yilmaz
Sritika Chakladar
Kyle Doud

Department of Computer Science and Software Engineering
Auburn University
3101 Shelby Center
Auburn, AL 36849, USA

Alice E. Smith
Alejandro Teran-Somohano

Department of Industrial and Systems Engineering
Auburn University
3301 Shelby Center
Auburn, AL 36849, USA

Halit Oğuztüzün
Sema Çam
Orcun Dayıbaş
Bilge K. Görür

Department of Computer Engineering
Middle East Technical University
Çankaya, Ankara 06800, Turkey

ABSTRACT

There are often concerns about the reliability of simulation results due to improper design of experiments, limited support in the execution and analysis of experiments, and lack of integrated computational frameworks for model learning through simulation experiments. Such issues result in flawed analysis as well as misdirected human and computational effort. We put forward a methodological basis, which aims to (1) explore the utility of viewing models as adaptive agents that mediate among domain theories, data, requirements, principles, and analogies, (2) underline the role of cognitive assistance for model discovery, experimentation, and evidence evaluation so as to differentiate between competing models and to attain a balance between model exploration and exploitation, and (3) examine strategies for explanatory justification of model assumptions via cognitive models that explicate coherence judgments.

1 INTRODUCTION

Despite successful automation of routine aspects of simulation and data management, scientific process continues to require considerable human expertise and effort (Honavar, Hill, and Yelick 2016). To extend the reach of human intellect on a broad range of tasks in scientific discovery, we promote the use of Model-Driven Engineering (MDE) strategies (Yilmaz, Chakladar, and Doud 2016) coupled with cognitive computing to address various tasks that are pertinent to simulation-driven discovery. These tasks include identifying, prioritizing, formulating questions, postulating mechanisms that serve as conjectures that explain the system or phenomena, defining or generating experiments designed to answer questions, drawing inferences, and evaluating results within an incremental and iterative discovery cycle.

The use of models as exploratory instruments that evolve via feedback into explanatory tools suggests that it is reasonable to view models as Dynamic Data-Driven Application Systems (Darema 2004). As a mediator

between theory and data (Morrison and Morgan 1999), models need to seek coherence in mechanistic hypotheses (e.g., behavioral rules) that govern a model's behavior. Figure 1 illustrates how models can play such mediation role between theory and data. The process starts with leveraging theoretical premises or domain-specific principles to construct hypotheses in the form of behavioral rules. The initial construction of the model is followed by simulation, which involves goal-directed experimentation with the model to explore its behavior over time (Yilmaz, Chakladar, and Doud 2016). Targeted instrumentation of the model results in simulation data that forms the basis for learning about the efficacy of hypothesized assumptions. The dynamic and mutually recursive feedback between the model and data facilitates adaptive learning (Bunge 1998, Gelfert 2016). That is, the data gathered through the simulation experiment is analyzed to make decisions about model's parameters, structure, and representation. Model revisions, if there is sufficient consensus, can result in theory revision. As a consequence, theoretical principles induced from limited data become increasingly accurate in their explanatory power.

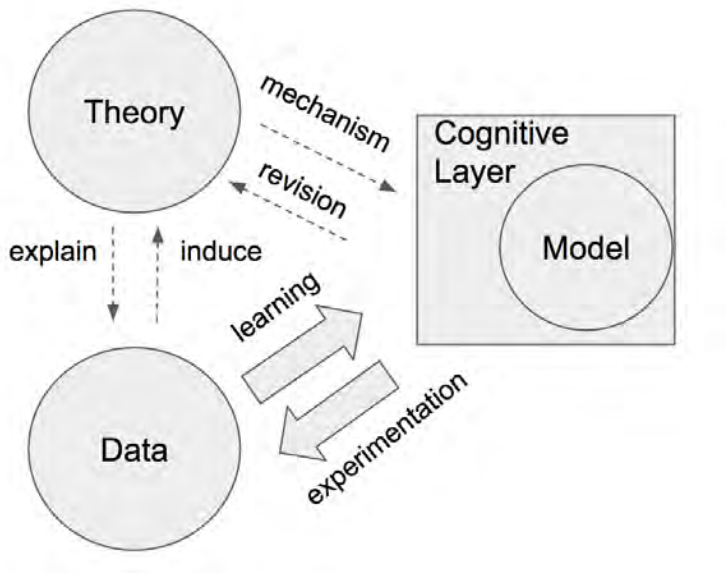


Figure 1: Models as mediators.

These observations motivate us toward developing a methodological foundation and a proof-of-concept model-driven cognitive system framework, which allows (1) exploring the utility of viewing models as adaptive agents that mediate among engineering domain theories, data, requirements, principles, and analogies, (2) understanding the role of cognitive assistance for model discovery, experimentation, and evidence evaluation so as to differentiate between competing system models, and (3) examining alternative strategies for explanatory justification of plausible models using an evolving run-time cognitive model that explicates coherence judgments. Such a foundation can be enabled by the provision of high-level Domain-Specific Languages (DSL) that can express and prioritize questions and generate experiments to address them. Alternative explanations (mechanisms) or expectations about how a system works (or should work) need to be characterized without considerations about their realization over a computational platform. To this end, computational models need to be synthesized by weaving presupposed mechanisms while experiment designs are generated from the questions and related hypotheses. Intelligent agents interpret experiments, orchestrate their execution, and, as learning takes place, by using a cognitive model, update the experiment model as well as the generative constructs (e.g., mechanisms).

In the rest of the paper, we provide the basic elements of the proposed framework. Following a brief overview of the underlying premises of our strategy in section 2, we introduce a taxonomy and a sketch of a model-guided process that allows mediation between theory and data during the model building phase. In

section 4, we examine the role and significance of Model-Driven Engineering principles in addressing the critical issues highlighted by the mediation process. To illustrate the use of these principles, the Domain-Specific Language, XPerimenter, is outlined in section 5, and to demonstrate the process of searching an experiment space, we discuss in section 6 how the design of experiments methodology can be coupled with optimization strategies to improve information gain efficacy while reducing computational time. Section 7 highlights the role of cognitive computing and formal methods to facilitate model adaptation. We conclude in section 8 with directions in furthering the research in model-driven computational discovery.

2 BACKGROUND

The adaptive nature of models as highlighted above is consistent with the "models as mediating instruments" perspective (Morrison and Morgan 1999). According to this view, a model needs to align theoretical and empirical constructs so as to converge to an explanation of expected behavior. This view is akin to the Dynamic Data-Driven Application System (DDDAS) perspective (Darema 2004), which promotes incorporation of on-line real-time data into simulation applications to improve the accuracy of analysis and the precision of predictions. As such, DDDAS aims to enhance applications by selectively imparting new observed data or derived knowledge into the running application so as to make it congruent with its context. The ability to guide measurement and instrumentation processes is critical when the measurement space is large. By focusing measurement in a subset of the observation space, the methodology reduces both the cost and the time to test, while also improving the quality and relevance of data. Viewing coupled model-experiment system dynamics as a DDDAS requires advancements in variability management, development of model interfaces to instrument the simulation for data collection, and facilities to incorporate data-driven inferences back into the model's technical space. The provision of run-time models and run-time dynamic model updating are critical features for closing the loop.

Our strategy is based on viewing models as *moderators*, *generators*, and *cognitive tools* (Gelfert 2016).

- **Models as moderators:** In the case of mediation, a model needs to align theoretical and evidential constructs so as to converge to an explanation of expected behavior. The moulding of the ingredients of the model to align theory and data suggests the autonomous characteristic of a model, measured as a function of the degree of *coherence* of the competing assumptions, mechanisms, and evidential observations.
- **Models as knowledge generators:** Models should not merely mediate existing theoretical and empirical elements. New elements and constructs need to be generated to not only conceptualize mechanisms, but also integrate them in ways that are logical and consistent.
- **Models as cognitive enablers:** Through the construction and manipulation of hypothetical features, models contribute as *epistemic tools* that facilitate generation and justification of models via experimentation. However, to facilitate acquisition of engineering knowledge, models should not only be connected with a valid and objective relation to the phenomena of interest, but also afford cognitive access to the information it contains.

3 MODEL-DRIVEN MEDIATION BETWEEN THEORY AND DATA

The role and utility of computational models as abstract and ideal representations are well acknowledged (Glotzer, Kim, Cummings, Deshmukh, Head-Gordon, Karniadakis, Petzold, Sagui, and Shinozuka 2009). As model development methodologies continue to reduce the gap between the solution and the problem space, scientists improve their ability to conceive abstractions necessary to address research questions. Reliable models are clearly necessary but not sufficient for successful application of simulations in addressing research problems (Kleijnen, Sanchez, Lucas, and Cioppa 2005). Proper design and management of experiments is critical to draw reliable conclusions from computer simulations.

Building on (Klahr and Simon 1999), the mediation strategy involves three main components that control the process from the initial formulation of hypotheses, through their experimental evaluation, to the

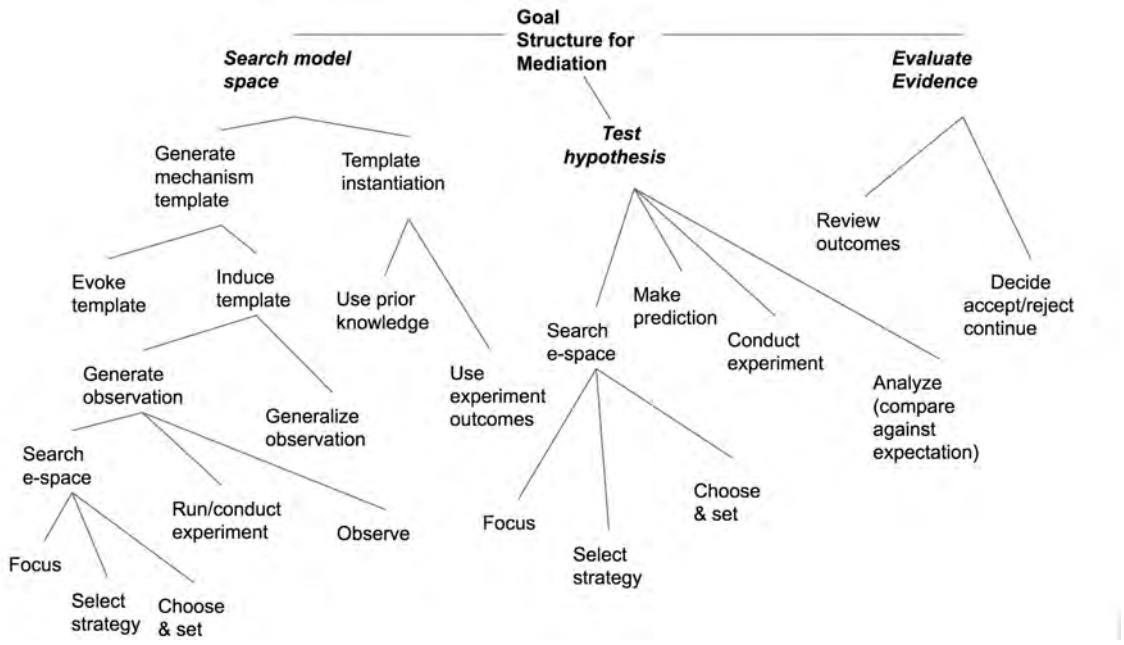


Figure 2: The goal structure of mediation.

decision that there is sufficient evidence to accept or reject hypotheses. The three major components are *Search Model Space*, *Search Experiment Space* (Test Hypothesis), and *Evaluate Evidence*. The output from the *Search Model Space* component is a set of fully-specified mechanistic models, which serves as input to the *Test Hypothesis* phase that involves simulated experiments, resulting in evidence for or against the hypothesis. The *Evaluate Evidence* component decides whether cumulative evidence warrants acceptance, rejection, or further consideration of the current hypothesis.

Search Model Space: Searching the model space requires identifying the general structure and the scope of the hypothesized model mechanism(s), followed by the refinement of the structure to make it instantiable. To generate model mechanisms, strategies for evoking or inducing a template are needed. Model-driven engineering principles and feature-oriented analysis (Meinicke, Thüm, Schröter, Krieter, Benduhn, Saake, and Leich 2016) provide a basis for generative modeling. Specifically, strategies that view models as algebras (Oliveira, Van Der Storm, Loh, and Cook 2013) allow flexible composition of models in terms of features. Evoking an existing template requires searching a frame to abstract away unnecessary imperative programmatic constructs, requiring the utilization of MDE strategies. When it is not possible to evoke a mechanism template, a new mechanism can be induced in the form of a metamodel by generating expected behavioral outcomes and by generalizing over the outcomes to induce a mechanism frame. The second issue that needs to be addressed is the template instantiation process, which takes as input a partially instantiated template and assign specific values to generate a fully-specified hypothesized model. Theory of template-based modeling provides a path for using prior knowledge or specific experimental outcome.

Search Experiment Space: This goal aims to generate an experiment that is appropriate for the current set of hypotheses being examined, to make a prediction by running the simulation experiment, and to match the outcome to expected behavior. Once the Search Experiment Space component produces an experiment, conducting the experiment involves execution of the simulation and may require distributing the replications across multiple machines to improve the performance. The *Analyze* (Compare) goal aims to provide a description of the discrepancy between the expected behavior and the actual outcome. The comparison involves both hypothesis testing and probabilistic model checking theory for evaluating the degree of satisfiability of abstract system properties. The outcomes from simulation replications are used

toward generating outcomes generalized as a Probabilistic Markov Model, which is verified against finite state verification patterns.

Evaluate Evidence: This component aims to determine whether or not the cumulative evidence warrant the acceptance or rejection of competing hypotheses. Various criteria can be used to evaluate evidence and hypotheses. These include plausibility, functionality, parsimony, etc. In the absence of hypotheses, experiments can be generated by intelligently navigating the experiment space. During the *Evaluate Evidence* phase, three general outcomes are possible. The current set of coherent hypotheses can be accepted, rejected, or considered further. In the first case, the discovery process stops. If the hypothesis is rejected, then the system returns to the *Search Hypothesis Space* phase. Accumulating evidences over hypothesized models need to be objectively scrutinized in a transparent and explainable manner. This is critical for instilling trust in the cognitive assistance system and its recommendations. In this context, the explanation model can serve dual purposes. Besides informing the user about the utility of competing model mechanisms, it can serve as a run-time model to guide the revision and selection of hypotheses, identifying which mechanisms to focus, and what experiments to generate to discriminate between competing models.

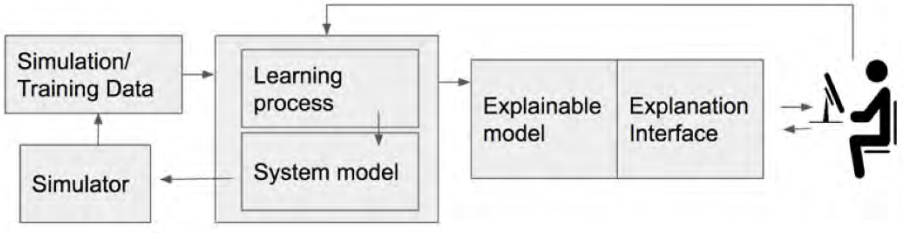


Figure 3: Explainable cognitive coherence model.

Explainable Cognitive Mediation: As shown in Figure 3, the target of explainable cognitive model is an end user who depends on decisions, recommendations, or actions generated by the modeling system, and therefore needs to understand the rationale for the system's decisions. For example, an engineer who receives recommendations needs to understand why the cognitive model has recommended certain hypothesized mechanisms or specific experiments over others. In seeking a balance between theoretical principles, evidence, system constraints, and mechanisms, a strategy is needed to attain a state of coherent justification that accounts for the recommended model mechanisms. The options available are deep learning explanation, interpretable models, and model induction.

4 ENABLING ADAPTIVE MEDIATION VIA GLOBAL MODEL MANAGEMENT

Model-Driven Engineering (MDE) (Völter, Stahl, Bettin, Haase, and Helsen 2013, Yilmaz 2015) has emerged as a methodology to manage complex simulation systems development by bringing model-centric thinking to the fore. The use of platform-independent domain models along with explicit transformation models facilitates deployment of simulations across a variety of platforms (Teran-Somohano, Smith, Ledet, Yilmaz, and Oğuztüzün 2015). While the utility of MDE principles in simulation development is now widely recognized, its benefits for experimentation have not yet received sufficient attention.

Global Model Management (GMM) is a Model Driven Engineering concept that aims to handle models, metamodels and their properties and relations in a model-engineering environment (Bézivin 2009). GMM is a sophisticated way of creating, storing, viewing, accessing, modifying, and using the information associated with all these modeling entities. In order to provide a model management for models within a model, the megamodel (Bézivin, Jouault, and Valduriez 2004) concept is introduced. A megamodel is a model that contains models and relations between them. The megamodel represents the Model Driven Engineering artifacts, including transformation composition and execution within a model (Vignaga, Jouault, Bastarrica, and Brunelière 2013). A GMM environment is useful to solve complex system management and model mediation issues, such as model consistency, model evolution and model sharing. For instance, in MDE

based projects, model to metamodel conformance must be established before model transformation takes place. If the conformance relations are lost, then inconsistencies occur and model transformation fails. GMM helps to avoid such inconsistencies and it is useful to achieve a uniform mechanism for process definition, variability, tailoring and evolution. For instance, in (Simmonds, Perovich, Bastarrica, and Silvestre 2015) automated process model tailoring utilizes megamodeling as the underlying formalism. In practice, experimenters deal with a variety of simulation models and GMM provides promising concept for their mediation as well as for numerous model management problems.

In our view, the experiment and simulation model spaces within a GMM should be loosely coupled to orchestrate the co-evolution of simulation and experiment spaces as learning takes place. For generating experiment specifications from research questions and hypotheses, the Design of Experiments (DOE) methodology (Kleijnen 2007, Montgomery 2006) provides a structured basis for automation. The DOE ontology defines the vocabulary and grammar, i.e., the abstract syntax for building the experiment domain model. To support the instantiation of the experiment specifications conforming to the DOE metamodel, a suitable DSL is developed. An experiment design can have various mandatory, alternative, and optional features, which are prominent attributes that facilitate modeling variants of experiments to support different objectives. The experiment model defined by the DSL is configured with the aspects specified by such an experiment feature model. In section 5, we briefly outline the Xperimenter DSL that illustrates variability management in experiment generation.

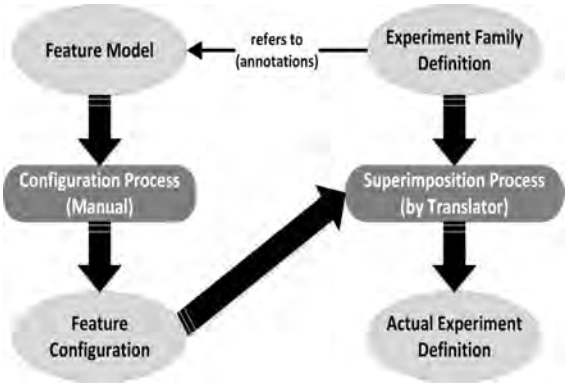
Experiment orchestration involves experiment design adaptation capabilities so that factors that are not significant in explaining the differences in the dependent variables are reclassified as control variables, and, if necessary, design schema can adapt as experimentation moves from variable screening to factor analysis. The aggregation of results for effective analysis and communication is a critical step. Regression trees, Bayesian networks, probabilistic model checking based on Markov models are effective ways of communicating which factors are most influential on the performance measures. Section 7 provides a brief overview of our efforts in adopting a formal method to analyze simulation data to assess the efficacy of the generated model in explaining desired behavioral properties.

5 XPERIMENTER: A DSL FOR VARIABILITY MANAGEMENT IN EXPERIMENTS

Xperimenter is a DSL for specifying simulation experiments. Being a declarative language, it allows domain scientists to specify an an experiment at a high level of abstraction. As such, it is independent of the host programming environment. Its translator generates the target environment entities as executable assets by interpreting the specifications. A typical target environment is a scientific workflow management system, such as Kepler. XPerimenter utilizes these assets to replicate the experiment by using a variability management mechanism to model and manage experiment variants. The fragments of an experiment specification are then mapped to the features, and the selected features are used to form a particular run of an experiment. Hence, the variability mechanism facilitates managed reuse of experiment assets.

As shown in Figure 4(a), domain and feature models are used to capture relevant aspects of simulation experiments. Our contribution is twofold: First, we defined a formal way to specify and execute an experiment to support replicability via semi-automated model transformations. Second, customized use cases for users with different skill levels are introduced. To our knowledge, it is the first attempt that brings together variability management with the design of experiment paradigm. Our approach supports systematic reuse by defining a formal strategy to reuse experiment assets.

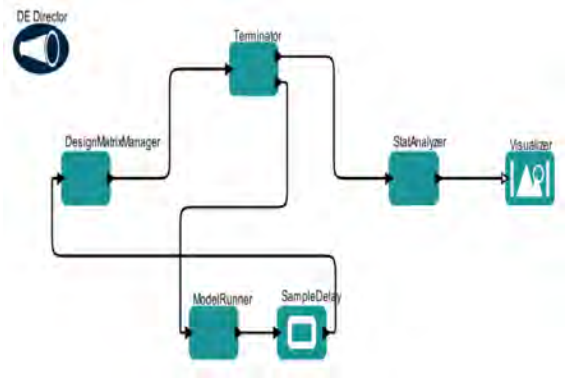
Although our approach is platform-neutral, the Kepler scientific workflow management environment is selected for demonstration purposes (Figure 4(c)). Kepler is an environment for the design and execution of scientific workflows, and it is based on the Ptolemy-II system which relies on the actor-oriented modeling paradigm that is based on two main entities: The *Director* component manages the data flow, and a set of *Actors* collectively execute the steps of the workflow. Although the experiment workflow is static, the actors can adapt in accordance with the feature configuration and DSL-based experiment design expressions. The *DesignMatrixManager* (DMM) is the actor that generates the sampling instances



(a) Experiment variability.

```
experiment QuadcopterExperiment {
        desc "Which PID parameter effects the controller quality
most";
        objective COMPARATIVE;
        design CompQuadcopterExpDesign;
        simulation QuadcopterSim;
        analysis AnovaAnalysis;
        visual DEFAULT;
        target KEPLER;
}
variable Ki : INTEGER group FACTOR [1, 10];
variable Kp : INTEGER group FACTOR [1, 10];
variable Kd : INTEGER group FACTOR [1, 10];
variable AV : INTEGER group RESPONSE;
design CompMachineIntExpDesign {
        method FULLFACTORIAL;
        varlist Ki Kp Kd AV;
}
simulation MachineSim {
        modelFile "c:\\quad_pid_sim.m";
        modelType DISCRETEEVENT;
        inport Ki : Ki;
        inport Kp : Kp;
        inport Kd : Kd;
        outport AV : AV;
}
analysis AnovaAnalysis {
        file "http://ceng.metu.edu.tr/~e1564178/xperimenter/anova-
service";
}
```

(b) Quadcopter experiment.



(c) Workflow execution.

```
experiment MachineInterference {
  desc "An experiment to find the best way to increase utilization of
    machines: buying a new machine or hiring a new repairman";
  objective COMPARATIVE;
  design CompMachineIntExpDesign;
  simulation MachineSim;
  analysis AnovaAnalysis;
  visual DEFAULT;
  timeout 180;
  target KEPLER;
}

variable M : INTEGER group FACTOR [3, 6];
variable R : INTEGER group FACTOR [1, 4];
variable UM : INTEGER group RESPONSE;

design CompMachineIntExpDesign {
  method FULLFACTORIAL;
  varlist M R UM;
}

simulation MachineSim {
  modelFile "c:\\machine_int.mdl";
  modelType DISCRETEEVENT;
  inport NumberOfMachines : M;
  inport NumberOfRepairmen : R;
  outport UtilOfMachines : UM;
}

analysis AnovaAnalysis {
  service "http://ceng.metu.edu.tr/e1564178/xperimenter/anova-service";
}
```

(d) Machine interference experiment.

Figure 4: Domain-specific language for experiment management.

(design points). The other function of the DMM is to accumulate and aggregate the results of the individual runs. The purpose of the *Terminator* component is to decide whether or not additional runs are necessary. As the name implies, the *ModelRunner* component executes the simulation model by using the provided parameters. Finally, the *StatAnalyzer* actor uses the statistical analysis method over the design matrix. In Kepler, discrete-event actors act only after they receive their inputs. The Xperimenter Environment (DSL grammar, Editor, Management Product, Kepler Translator etc.) is under continuous maintenance and development. The current prototype is accessible at the following source code repository: `https://github.com/odayibas/xperimenter`.

XPerimenter Workflow – The true user (non-programmer) and power user (code-savvy user) terms are used to distinguish different roles and use cases associated with a scientific workflow system. In terms of experiment management, true users are expected to use a higher level tool that is enhanced by the variability model, and power users have more flexibility by using the syntax of the domain-specific language. The value of transforming high level specifications into executable experiments is twofold: First, step-wise refinement engenders different level of interactions. Second, the generation of the boilerplate code makes polyglot simulation experiments possible. The users are not bound to a single execution environment; that is, it is possible to have a simulation model in one language and analysis code in another one by using our approach.

The experiment definition involves a reference to an experiment design and each design is given a name, a sampling method to be used, and a list of variables. The DSL assumes that the *modelFile* is a runnable entity in the target platform in order to provide a platform independent way to execute the simulation model. Additional information such as the *modelType* is optional, but it can be used for prospective target

platforms. The aforementioned design entity involves a set of variables. The concrete definition of the variable covers its classification and its high/low values. It is also possible to attach a pseudo random number generator (generator) to the variables to introduce randomness in stochastic models. These features facilitate connecting the model's technical space to the technical space of the experiments. Next, we discuss the search process within the experiment space.

6 SEARCHING THE EXPERIMENT SPACE

Space-filling designs have been widely used to efficiently cover the experiment space to reveal the impact and significance of independent factors on the dependent variables under controlled settings. In system evaluation and comparisons, there are often two or more system configurations, and the objective is to determine if the difference between measured outcomes are statistically significant. To this end, in the search of the experiment space, the goal is to uncover the causation of the differences between two or more models using identical set of decision parameters. Our approach involves developing a smart search technique that integrates design of experiments and optimization modules to identify the gap between simulation results.

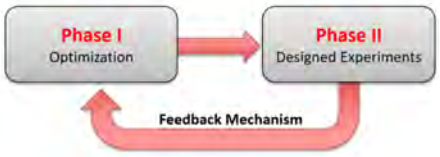
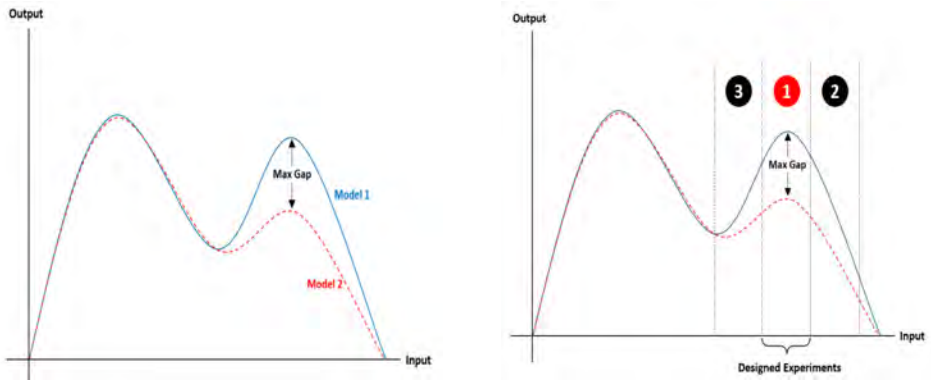


Figure 5: Feedback between experiment design and optimization.

Specifically, *Model Alignment* is defined as the process of determining whether two models can produce the same results. As shown in Figure 5 our approach is a two-phased methodology:

- Phase I: Optimization (Search) – Use Particle Swarm Optimization to explore the response surface for the regions of interest that have a potential gap between the model response surfaces.
- Phase II: Designed Experiments – Perform statistically designed experiment over the region of interest to determine whether any significant differences exist.



(a) Optimization Phase (b) Design of Experiments

Figure 6: Two-phased exploration of the response surface.

This unique integration of experiment design, outcome analysis, and optimization is a powerful and general approach to accomplish difficult tasks in simulation validation, comparison, and reuse. To this end,

we are developing a general purpose optimization library rather than a collection of heuristic algorithms. The result is a simplified smart search language that integrates design of experiments and optimization (see Figure 6(a). Since the current model alignment approaches are limited to a restricted set of parameter value combinations, we introduced a new intelligent methodology to determine the equality or difference between models over possible parameter value combinations without using an exhaustive search. Our approach enables analysts to characterize the outcomes of different simulation models, and has applications in model transformation verification, as well as model comparison.

As shown in Figure 6(a), the objective function in the optimization phase is to maximize the absolute differences in responses between the two models (e.g., two different quad-copter controllers with three input gain parameters and the total linear distance travelled output parameter). Specifically, we used *Particle Swarm Optimization* to find a region of potential difference (max gap). In the second phase, we create a designed experiment in the optimal region. Among the type of designs tested are the factorial and hypercube designs. Intelligent agent components are used to utilized to apply expert rules to recommend actions to the user based on Kolmogorov-Smirnov test for two empirical distributions: Chi-squared for goodness of fit and Anderson-Darling test for tail-fitting, bias, variance, and expected values.

7 MODELS AS COGNITIVE AGENTS

As an active entity, models with cognitive capabilities will provide features that overlay the base model and augment it to support the mediation process. In this view, an autonomous model is construed as an ensemble of models that evolve as learning via experimentation takes place. As such, we posit that models need to be designed with variability management in mind to support customization to address a variety of experiment objectives, especially when the target system has multiple facets.

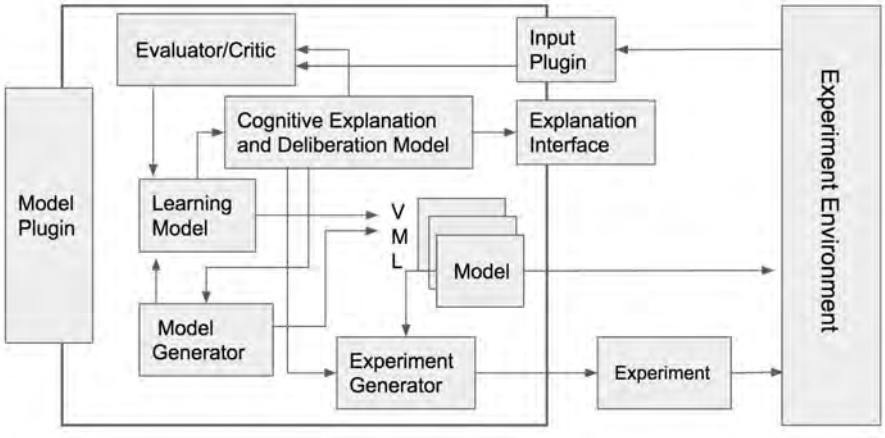


Figure 7: Models facilitate mediation between theory and data.

Figure 7 illustrates the building blocks of the proposed active model concept that is coupled to an experimentation environment to maintain a mutually beneficial and adaptive relation between domain theory and data. Theoretical constructs are characterized by the *Variability Management Layer* (VML) and the models that encapsulate the mechanistic hypotheses, principles, and constructs underlying the theory. A family of models are defined in terms of features that can be configured to synthesize alternative models. Features specify mandatory or optional standard building blocks of a family of systems along with their interrelationships. Variability management via feature models is a strategy used in product-line engineering to automate the initial construction of products. However, in our context, the variability management subsystem does not only enable the specification of a variability model, but also support its run-time adaptation to posit model updates in the host simulation modeling language. The selection of variants in

the form of features triggers specific actions that customize the model by adding, removing, or updating functional units in the target model at run-time.

The input plug-in shown Figure 7 transforms raw experiment data into a format that is used for analysis by the *Evaluator/Critic* component. The *evaluator* can be construed as simple as a filter or a functional map that further abstracts the data. However, to provide cognitive assistance, a sophisticated model needs to facilitate the use of formal methods such as *Probabilistic Model Checking* to determine the extent to which an expected regularity is supported by the simulation data. The evaluator performs hypothesis testing to discern the support of model mechanisms to the phenomenological hypotheses under consideration. Those mechanisms that provide statistically significant support to the evidence or expected behavior are retained, while others are revised or declined from further consideration. The learning model uses the results of the evaluator to update the confidence levels over the competing hypotheses.

Formal methods of software verification are traditionally used to prove that algorithms are correct, in the sense that they exhibit the mechanisms that are necessary to output the correct value(s). Based on this observation, the connection can be made that, if formal verification can be done autonomously, without the intervention of human analysts, it could be used to verify mechanistic hypotheses on simulation models at a faster rate. Applying formal methods to hypothesis evaluation would allow not only evaluating a mechanism when it is in conflict with the evidence from prior experimentation, but for exploration of new hypotheses that increase information gain as well. Probabilistic model checking is a means of verifying mechanistic hypotheses autonomously from recorded experiment executions. The aim of this is to take some of the guesswork out of experimentation in a systematic way that reduces time spent on designing experiments, and reducing costs of experiments that reveal less information.

This goal is accomplished by a process of weaving data logging code into simulation models with code generated by an aspect-oriented programming extension (AspectJ) as a method of instrumentation. This allows a scientist to record quantitative observations without additional refinement of the model to accommodate data logging, and use the recorded data to form a verification model for automated hypothesis testing. A domain specific language allows a user who has less than expert programming experience, but advanced domain experience to carry out experiments in a simple and straightforward manner. Hypotheses in this DSL can be defined in terms of specification patterns, which is a high-level language abstraction to describe well-defined temporal logic properties that can be checked against the produced verification model. Integrating the results of the probabilistic model checking into an explanatory coherence network of hypotheses and evidence provides an easy-to-interpret visual representation of the body of domain knowledge, and can guide a scientist to the next step of experimentation.

8 CONCLUSIONS

The strategy presented herein can be extended in several ways. First, new features can provide fine-grained variants, which create a more flexible environment. Second, alternative translators allow targeting platforms other than Kepler. Also, a provenance mechanism based on the proposed domain model is desirable to store information about conducted experiments and to deduce new search directions based on prior experiments. Thus, defining and adopting a formal way to support provenance is of interest. Our research on integrating cognitive computing into experiment execution and evaluation will also continue. Specifically, we are implementing a Bayesian Net strategy to evaluate the statistical significance of hypotheses (i.e., behavioral mechanisms) in explaining evidence or expected behaviors, which are represented as propositions. To facilitate explanation of the learned model, the Bayesian Net can be mapped onto an explanatory coherence model to signify which behavioral mechanisms cohere and work together to provide robust solutions toward attaining desired behavioral objectives.

REFERENCES

Bézivin, J. 2009. "If MDE is the Solution, then What is the Problem?". In *International Conference on Software Language Engineering*, 2–2. Springer.

Bézivin, J., F. Jouault, and P. Valduriez. 2004. "On the Need for Megamodels". In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

Bunge, M. 1998. *Philosophy of Science: From Problem to Theory*, Volume 1. Transaction Publishers.

Darema, F. 2004. "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements". *International Conference on Computational Science. Lecture Notes in Computer Science (LNCS)* 3038:662–669.

Gelfert, A. 2016. *How to Do Science with Models*. Springer Briefs in Philosophy. Cham: Springer International Publishing.

Glotzer, S., S. Kim, P. Cummings, A. Deshmukh, M. Head-Gordon, G. Karniadakis, L. Petzold, C. Sagui, and M. Shinozuka. 2009. *Research and Development in Simulation-based Engineering and Science*.

Honavar, V. G., M. D. Hill, and K. Yelick. 2016. "Accelerating Science: A Computing Agenda". Technical report, Computing Research Association.

Klahr, D., and H. a. Simon. 1999. "Studies of Scientific Discovery: Complementary Approaches and Convergent Findings.". *Psychological Bulletin* 125 (5): 524–543.

Kleijnen, J. P. C. 2007. "Design and Analysis of Simulation Experiments".

Kleijnen, J. P. C., S. M. Sanchez, T. W. Lucas, and T. M. Cioppa. 2005, aug. "State-of-the-Art Review: A Users Guide to the Brave New World of Designing Simulation Experiments". *INFORMS Journal on Computing* 17 (3): 263–289.

Meinicke, J., T. Thüm, R. Schröter, S. Krieter, F. Benduhn, G. Saake, and T. Leich. 2016. "FeatureIDE: Taming the Preprocessor Wilderness". In *Proceedings of the 38th International Conference on Software Engineering Companion*, 629–632. ACM.

Montgomery, D. C. 2006. *Design and Analysis of Experiments*. John Wiley and Sons.

Morrison, M., and M. S. Morgan. 1999. "Models as Mediating Instruments". *Ideas in context* 52:10–37.

Oliveira, B. C. d. S., T. Van Der Storm, A. Loh, and W. R. Cook. 2013. "Feature-Oriented Programming with Object Algebras". In *European Conference on Object-Oriented Programming*, 27–51. Springer.

Simmonds, J., D. Perovich, M. C. Bastarrica, and L. Silvestre. 2015. "A Megamodel for Software Process Line Modeling and Evolution". In *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*, 406–415. IEEE.

Teran-Somohano, A., A. E. Smith, J. Ledet, L. Yilmaz, and H. Oğuztüzün. 2015. "A Model-Driven Engineering Approach to Simulation Experiment Design and Execution". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, I. Moon, W. K. Chan, and T. Roeder, 2632–2643. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Vignaga, A., F. Jouault, M. C. Bastarrica, and H. Brunelière. 2013. "Typing Artifacts in Megamodeling". *Software & Systems Modeling* 12 (1): 105–119.

Völter, M., T. Stahl, J. Bettin, A. Haase, and S. Helsen. 2013. *Model-driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

Yilmaz, L. 2015. "Toward Agent-Assisted and Agent-Monitored Model-Driven Simulation Engineering". *Concepts and Methodologies for Modeling and Simulation*:3–18.

Yilmaz, L., S. Chakladar, and K. Doud. 2016. "The Goal-Hypothesis-Experiment Framework: A Generative Cognitive Domain Architecture for Simulation Experiment Management". In *Proceedings of the 2016 Winter Simulation Conference*, edited by P. Frazier, T. Roeder, R. Szechtman, and E. Zhou, 1001–1012. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

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. His research interests are in agent-directed simulation, cognitive computing, and model-driven science and engineering for complex adaptive systems. He is the former Editor-in-Chief of Simulation: Transactions of the Society for Modeling and Simulation International and the founding organizer and general chair of the Agent–Directed Simulation Conference series. His email address is yilmaz@auburn.edu.

SRITIKA CHAKLADAR is a Graduate Student at the Department of Computer Science and Software Engineering in Auburn University. Her research interests are in model-driven engineering and computational biomedical research. Her email address is szc0098@auburn.edu.

KYLE DOUD is a Graduate Student at the Department of Computer Science and Software Engineering in Auburn University. His research interests are in model-driven engineering and formal methods. His email address is krd0015@auburn.edu.

ALICE E. SMITH is is the Joe W. Forehand/Accenture 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 $2,000 ISI Web of Science citations and has been a principal investigator on projects with funding totaling over $6 million. She is an area editor of INFORMS Journal on Computing and Computers & Operations Research and an associated editor of IEEE Transactions on Evolutionary Computation. Her email address is smithae@auburn.edu

ALEJANDRO TERAN-SOMOHANO is a PhD Candidate in Industrial and Systems Engineering at Auburn University. He holds a Bachelors degree in Computer Engineering from the Instituto Tecnologico Autonomo de Mexico (ITAM) and M.S. in Industrial Engineering from Auburn University. His email address is ateran@auburn.edu.

HALİT OĞUZTÜZÜN is a 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, and PhD in Computer Science from University of Iowa. His research interests include model-driven engineering and distributed simulation. His email address is oguztuzn@ceng.metu.edu.tr.

SEMA ÇAM is a PhD Candidate in Computer Engineering at Middle East Technical University (METU). She holds a Bachelor's degree in Computer Engineering. Her email address is cam@ceng.metu.edu.tr.

ORÇUN DAYIBAŞ is a PhD Candidate in Computer Engineering at Middle East Technical University (METU). He holds a Bachelor's degree in Computer Engineering from Hacettepe University and M.Sc. in Computer Engineering from METU. His email address is orcun.dayibas@metu.edu.tr.

B. KAAN GÖRÜR is a computer engineer at Roketsan A.S. in Ankara, Turkey. He is also a PhD student in Department of Computer Engineering in Hacettepe University. His research interests are parallel and distributed simulation, agent based modeling and simulation, model driven engineering, and simulation visualization. His email address is bkaangorur@gmail.com.