MODEL-DRIVEN ENGINEERING FRAMEWORK FOR REPLICABLE SIMULATION EXPERIMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ORÇUN DAYIBAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JULY 2019

Approval of the thesis:

MODEL-DRIVEN ENGINEERING FRAMEWORK FOR REPLICABLE SIMULATION EXPERIMENTS

submitted by **ORÇUN DAYIBAŞ** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar Dean, Graduate School of **Natural and Applied Sciences**	______________
Prof. Dr. Halit Oğuztüzün Head of Department, **Computer Engineering**	______________
Prof. Dr. Halit Oğuztüzün Supervisor, **Computer Engineering Department, METU**	______________
Prof. Dr. Levent Yılmaz Co-supervisor, **Computer Science and Software Eng. Dept., Samuel Ginn College of Engineering, Auburn University**	______________

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu Computer Engineering Department, METU	______________
Prof. Dr. Halit Oğuztüzün Computer Engineering Department, METU	______________
Prof. Dr. Tolga Can Computer Engineering Department, METU	______________
Assoc. Prof. Dr. Kayhan İmre Computer Engineering Department, Hacettepe University	______________
Assist. Prof. Dr. Hacer Yalım Keleş Computer Engineering Department, Ankara University	______________

Date:

ABSTRACT

MODEL-DRIVEN ENGINEERING FRAMEWORK FOR REPLICABLE SIMULATION EXPERIMENTS

Dayıbaş, Orçun
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Halit Oğuztüzün
Co-Supervisor: Prof. Dr. Levent Yılmaz

July 2019, 63 pages

Simulation experiments allow the user to capture the specific variability of multiple interdependent processes. The use of simulation models is widely accepted by practitioners from diverse areas of applied sciences. Therefore, simulation experiments are an essential part of computational science and engineering. Nevertheless, simulations are rarely replicated due to reuse and maintenance challenges related to models and data. In this respect, it is proposed that some crucial and labor intensive parts of the simulation experiments could be replaced or supported by model transformations. This work focuses on model-driven engineering practices to enable replicable and reusable simulation experiments. These practices are used to devote researchers' time to analyze the system under investigation rather than dealing with low level details to create a working environment. The results of our framework development work and our research directions are presented.

Keywords: Design of Experiment, Simulation Experiment, Variability Management,

ÖZ

TEKRARLANABİLİR SİMÜLASYON DENEYLERİ İÇİN MODEL TABANLI MÜHENDİSLİK ÇERÇEVESİ

Dayıbaş, Orçun
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Halit Oğuztüzün
Ortak Tez Yöneticisi: Prof. Dr. Levent Yılmaz

Temmuz 2019 , 63 sayfa

Simülasyon deneyleri, kullanıcıya birden fazla bir birine bağımlı sürecin belirli değişkenliklerini yakalama olanağı sağlar. Simülasyon deneylerinin kullanımı, uygulamalı bilimlerin değişik alanlarındaki uygulayıcılar tarafından geniş kabul görmüştür. Bu nedenle, simülasyon deneyleri bilişsel bilimler ve mühendisliğin önemli bir parçasıdır. Diğer taraftan, simülasyonlar, model ve verilere ilişkin yeniden kullanım ve idame zorlukları nedeniyle ender olarak tekrarlanabilirler. Bu bağlamda, simülasyon deneylerinin bazı önemli ve emek yoğun kısımlarının, model dönüşümleri ile tamamen veya kısmen ele alınmasına dayalı bir yöntem önerilmektedir. Bu çalışma tekrarlanabilir, yeniden kullanılabilir simülasyon deneylerini olanaklı kılmayı kolaylaştırmak adına model-güdülü mühendislik pratiklerine odaklanmaktadır. Bu pratikler, araştırmacıların zamanlarını çalışma ortamının yaratılması gibi alt seviye detaylar ile uğraşmak yerine incelenen sisteme ilişkin analizlere ayırması için kullanılmaktadır.

Anahtar Kelimeler: Deney Tasarımı, Simülasyon Deneyi, Değişkenlik Yönetemi, Mo-

del Tabanlı Mühendislik

To my family...

ACKNOWLEDGMENTS

I would like to thank all members of MDA research group (the both Auburn University and METU teams). It was a pleasure to work with each one of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ANCOVA	Analysis of Covariance
ANOVA	Analysis of Variance
CSV	Comma Separated Value
CIM	Computation Independent Model
DeMO	An Ontology for Discrete-event Modeling and Simulation
DOE	Design of Experiment
DSL	Domain-Specific Language
EMP	Eclipse Modeling Project
EDA	Exploratory Data Analysis
EXPO	An Ontology of Scientific Experiments
FODA	Feature-Oriented Domain Analysis
LCG	Linear Congruential Generator
M2T	Model-to-Text Transformation
M&S	Modeling and Simulation
MANCOVA	Multivariate Analysis of Covariance
MANOVA	Multivariate Analysis of Variance
MDA	Model-driven Architecture
MDD	Model-driven Development
MDE	Model-driven Engineering
PD	Proportional Derivative
PID	Proportional Integral Derivative
PIM	Platform Independent Model
PL	Programming Languages
PRNG	Pseudorandom Number Generation

PSM	Platform-specific Model
RSM	Response Surface Methodology
SED-ML	Simulation Experiment Description Markup Language
SESSL	Simulation Experiment Specification via a Scala Layer
SPL	Software Product Line

CHAPTER 1

INTRODUCTION

1.1 The Scientific Method and Experimentation

Scientists use the "scientific method" to develop their knowledge about the world. The attained knowledge may form public domain intellectual assets or used to lead new explorations as per the results. The scientific method can be considered as a common sense while looking at the world to understand rather than an academic dictation. Every exploratory study boils down to same steps and these steps forms a cycle (refer to Figure 1.1). Basically, observation leads to a hypothesis (inductive reasoning) and the scientist makes a reasonable prediction based on that explanation.

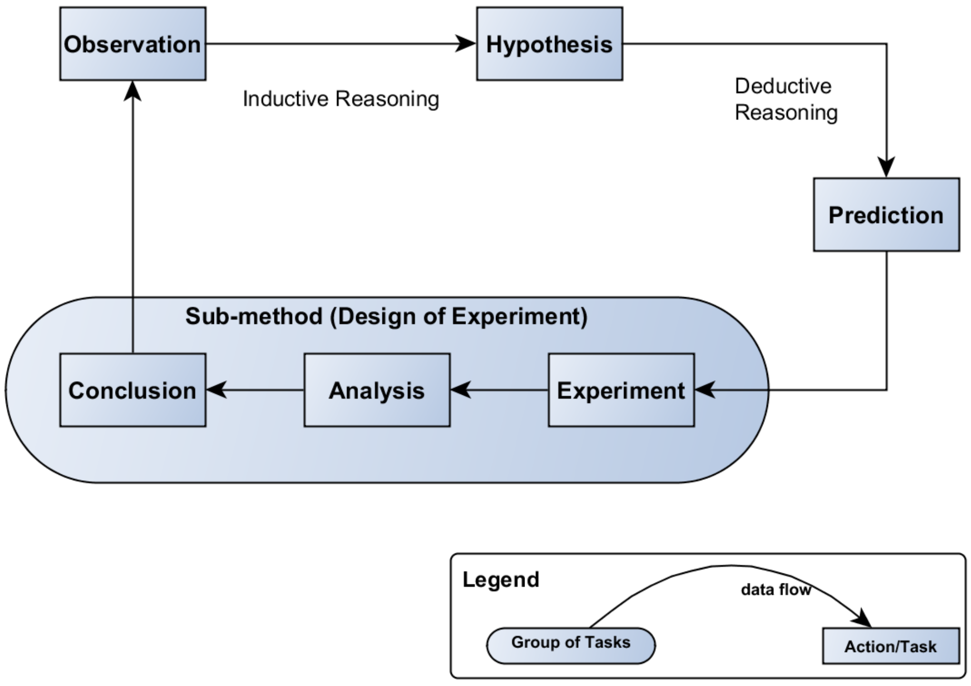


Figure 1.1: The Scientific Method

The most important characteristic of that explanation is to be testable by experimentation (possibly conducted by other parties). Being testable is a vital point because it is the essence of the scientific method to ensure that the foundation of the knowledge is stable and objective. These strong and testable pillars are used by practitioners under the shade of rational skepticism. Therefore, objective and reproducible experiment results are needed and "DOE - Design of Experiment" (Experimental design) is a process to create and conduct robust, reliable and efficient experiments. DOE defines a repeatable way and consolidates the best practices in this respect.

Experimental design (DOE) traditionally refers to physical experiments originating in laboratory experiments (physical models). However, at least parts of these experiments are good candidates to be handled in a software environment (mathematical models). Therefore, use of computer simulations is a paradigm shift for some labor-intensive experiments and orders of magnitude less time, cost and effort is expected by using a simulation model. Figure 1.2 depicts the process of data science in general. First, the both of computer simulations and physical experiments are used to test hypothesis in modern data science projects. Another important characteristic of the process is to have inherent cycles and some alternative paths. For instance researchers may update their hypothesis and pivot the experiment into slightly different version of it and this case can be repeated several times for even only one experiment. It is also not an extreme that researchers use physical and mathematical models in conjunction for a single research question. Thus, we need to manage the whole process effectively since the experimentation system evolves over the course of the life-cycle of the research and it is also contributed by different parties other than one single person.

In terms of computer simulations Exploratory Data Analysis (EDA) is also important sub-process because it is very first step to produce the mathematical models which are directly utilized in the simulations. The practitioners also make sense of the data and figure out what questions to ask in this analysis step. It is important to look for patterns, trends and outliers to create reliable models and/or algorithms. Quality of simulations and experiments are directly affected by quality of the process as a whole. Therefore, the success depends on implementation of the processes utilizing best software engineering practices with proper tool support.

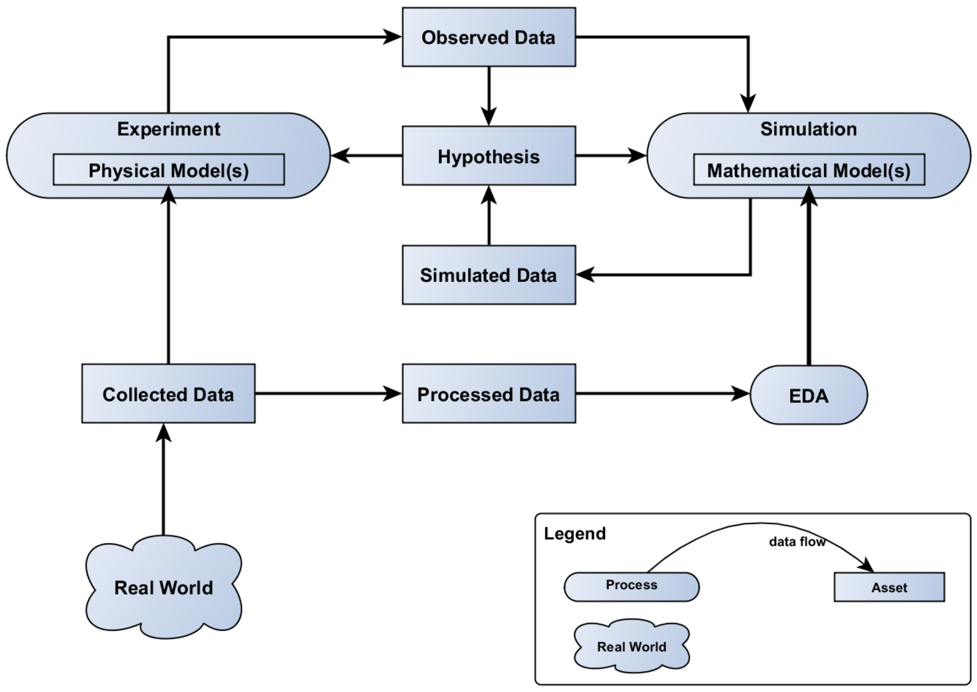


Figure 1.2: Extended Scientific Method and Simulation Experiments

1.2 Model-driven Experiment Management and Simulations

Model-driven engineering (MDE) is a design and development approach for development of complex systems (e.g. software-intensive systems). MDE is an approach to create complex system where models are the principal elements of the development process rather than implementation level artifacts only. It defines system functionality using platform independent/specific models (respectively CIM, PIM and PSM). Furthermore, transformations are main mechanism to generate artifacts at different levels of abstraction (see Figure 1.3). We propose to support the experimentation process by using the model-driven engineering approach (refer to Chapter 5 for details). In this respect, we are introducing a domain specific language (DSL) and its environment to manage simulation experiments. We need to elaborate our scope since “management” is a very broad term, before giving other details about the study.

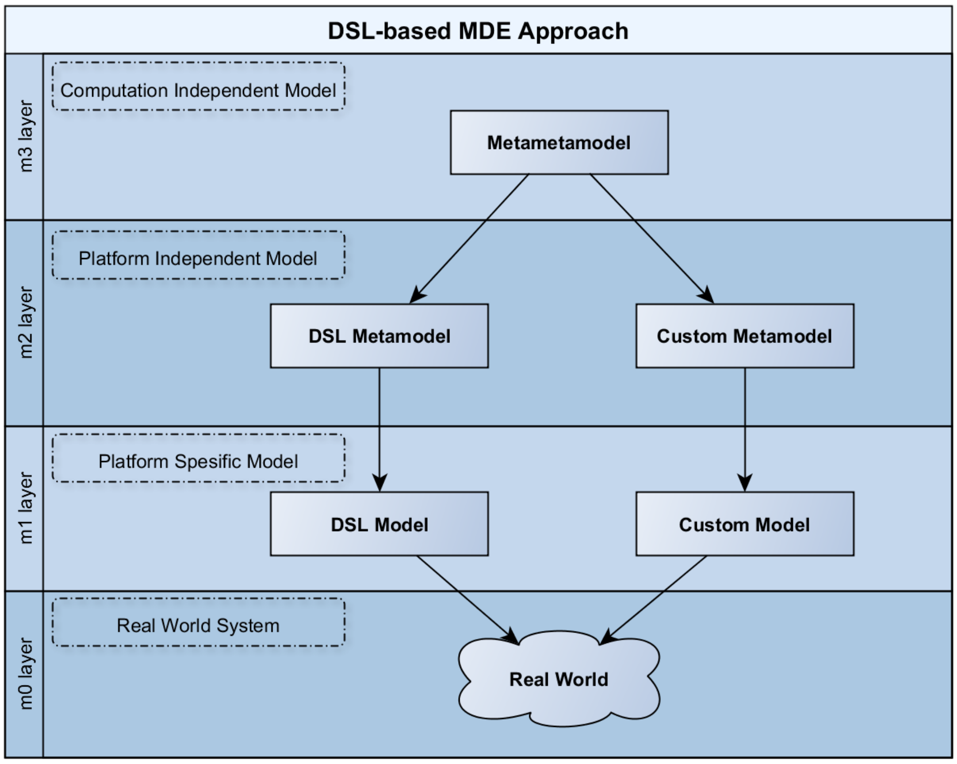


Figure 1.3: DSL-based Model-driven Engineering Approach (arrows depict "represents to" relations)

Experiments consist basically of a subject (an information source) and its environmental conditions (an experimental setup). Simulation experiments use system models as an information source differently from physical ones. Zeigler coined the "experimental frame" term to isolate specific interaction for the both real system and its model [3]. In terms of the simulation experiments, the separation between the system model and the experimental frame allows us to apply the same experimental frame to different system models and vice versa. In this respect, conducting experiments as-is (replication) means to run the system model with the experimental frame of the originator. It is a way of cross-checking to instill trust in the experiment results. The replication covers all experiment assets, including its model and data. Furthermore, reuse is a way to use assets (the artifacts of experiment workflow) in the context of another experiment. Reuse is a fundamental way to facilitate the creation of new experiments; moreover closely related experiments may form a family and that family can be managed to increase productivity. Although sometimes the terms are interchangeably used, there is a clear distinction between “replication” and “reproducibility” [4]. Reproducibility is a matter of verification, leading to same scientific results using a data set other than that of the originator (Table 1.1 helps to understand these terms and their use in our context).

Table 1.1: Replication < Reproduction < Reuse

	Model	**Data**	**Context**
Replication	Same	Same	Same
Reproduction	Same / Different	Same / Different	Same
Reuse	Same / Different	Same / Different	Same / Different

The main objective of this work is to improve the experimentation process by using model-driven development strategies. The strategy was introduced informally in [5]. However, formalization (in the form of metamodeling) is necessary to leverage available technologies, particularly, model-driven engineering. As per model-driven engineering approach, creating formal models and defining transition among these different abstraction level of artifacts is the primary strategy. The core contribution of this work is to define these transitions and models to validate applicability of the concept.

It is acknowledged that the part of this work was published as a journal article in the “Journal of Simulation” with title of “On the use of model-driven engineering principles for the management of simulation experiments” (Please refer to the Chapter 7 for the detailed information about the publication).

The rest of the dissertation is structured as follows: Chapter 2 presents an overview of the state of the art along with comparisons with our work in modeling simulation experiments. Our proposed approach is elaborated in Chapter 3. Chapter 4 provides the details of the Domain-Specific Language, named Xperimenter, for experiment modeling and Chapter 5 describes a generative method to produce executable experiments and manage their life cycle. Three case studies demonstrate the approach in Chapter 6. The final section concludes our proposed model-driven methods for effective simulation experiments and points to future research directions.

CHAPTER 2

STATE OF THE ART

There are several approaches in the literature that aim to increase productivity of simulation experimentation process. These approaches handle the issue at a variety of levels. As expected, higher level approaches such as creating general-purpose simulation tools for specific domains naturally have a higher impact on productivity [6] at the cost of decreased reuse. These tools are more practical to be used for the very fixed contexts and in general the repeatability is supported by adopting this approach solely. On the other hand, reuse of the assets such as simulation models has also remarkable effects on overall productivity of the process [7, 8]. Furthermore, more traceable process with reused assets increases the success of the research and that increased transparency also contributes the scientific respect of the simulation [9].

In this respect, we use scientific workflow engines to execute the whole simulation experimentation process. The present implementation of our approach relies on a specific scientific workflow engine, namely, Kepler [10], to encapsulate the simulation models. The reader can refer to [11] for a presentation of some other well known scientific workflow management systems and their comparison with Kepler. It is worth to highlight that our approach is not restricted to Kepler, it is just the very first implementation of many possible target environments (please refer to Chapter 5 for the details). Nevertheless, ad-hoc reuse of the experiment assets does not satisfy the true users (these users are experts in their scientific field of study, but with hardly any programming skills. The real world studies are generally conducted with a cross-disciplinary team and that diversity must be addressed by the adapted framework and/or methodology [12]. Please refer to [13] for the details of the true/power user distinction if you feel it is not clear because that terminology is used extensively through the rest of this dissertation in the context they provided.

In order to provide a systematic way to manage reusable assets, we need a well established method. In this respect, our approach facilitates managed reuse by using variability management techniques. Czarnecki *et al.* compare different approaches on that subject in [14]. As per their comparison, feature modeling is the most widely and most diversely used method by far. Furthermore, feature models can be directly associated with domain models [15]. We use features to encapsulate the requirements of experimenters.

Sturrock argues that a detailed functional specification of a simulation experiment plays key role in a simulation study [16]. Moreover, [17] advises to have this specification before we start modeling to avoid unexpected costs. As a functional specification, our DSL can be seen an adequate medium to share information among the practitioners. Sargent recommends an eight-step procedure in model verification and validation [18]. Several steps require coordination of different parties through exchange of the specification and periodic runs of the model. Furthermore, simulation modeling requires many experimental frames to be used to validate the model. Our framework adopts DOE method [19] in this respect. In [20], the authors give useful experimental designs and the benefits of this statistical approach in terms of cost and effort.

In [21], the authors identify “ensuring engineering reproducibility to enable the re-execution of analyses, and the replication of results” as a key challenge in scientific workflows. Furthermore, many simulation specific parameters (number of runs, random number allocations, time management issues, etc...) complicate the challenge for simulation experiments [22]. Such complications underline the importance of managed experimentation to reuse existing know-how on simulation experiment designs. Our approach formalizes the whole process of simulation experiment to provide replicable results. This holistic approach takes advantage of using model-driven engineering practices (e.g. model transformations [23]) to enhance replicability.

SED-ML is a proposed standard for encoding simulation setups. The referenced (Level 1 Version 1) version of the standard does not allow nesting or ordering of tasks. In [24], the authors state that modelers would provide SED-ML scripts with their manuscript submission that defines the detailed steps of the execution. This approach might be considered compatible with ours. It is possible to integrate our experiment definition to SED-ML by defining a model-to-text (M2T) transformation.

There exist a few ontologies like DeMO [25] to formalize the modeling part of the simulation experiments, but their main focus is to create a data interchange format for the simulation models. We rely on a scientific workflow engine to encapsulate the simulation model but we can still utilize DeMO, or other ontologies; our work is complementary to theirs. EXPO [26] is an ontology of scientific experiments and our formal models are influenced by that work. To the best of our knowledge, EXPO is the closest formalization effort to our approach. Nevertheless, our scope is restricted to computational experiments and only a part of their generic ontology overlaps ours partially.

SESSL is a domain-specific language for simulation experiments [27]. In terms of facilitating the use of a DSL to specify the simulation experiments, SESSL is a recent and comparable approach to our work. SESSL is an internal DSL while Xperimenter is an external one and SESSL does not explicitly support the notion of Design of Experiments. It does not have managed methods to decide experiment design points, the user is free to set any value to create experimental runs. Therefore, being an external DSL, portability of Xperimenter is not bounded by any host language. It is also easier to use by true users with its variability management support. Our work can be regarded as an attempt to answer the authors' call for a DSL for "experimenting with formal models (and possibly generating the corresponding workflows)" in [28].

"Open Knowledge Maps" is a relatively new but very useful tool to identify relevant concepts. Figure 2.1 is an overview of key research topics (MDE, DOE, DSL, Computer Simulation) of this work and their relations. It is simplified and aggregated version of the related maps on Open Knowledge Maps website [29, 30, 31, 32]. It can be used to get an overview of our research topics.

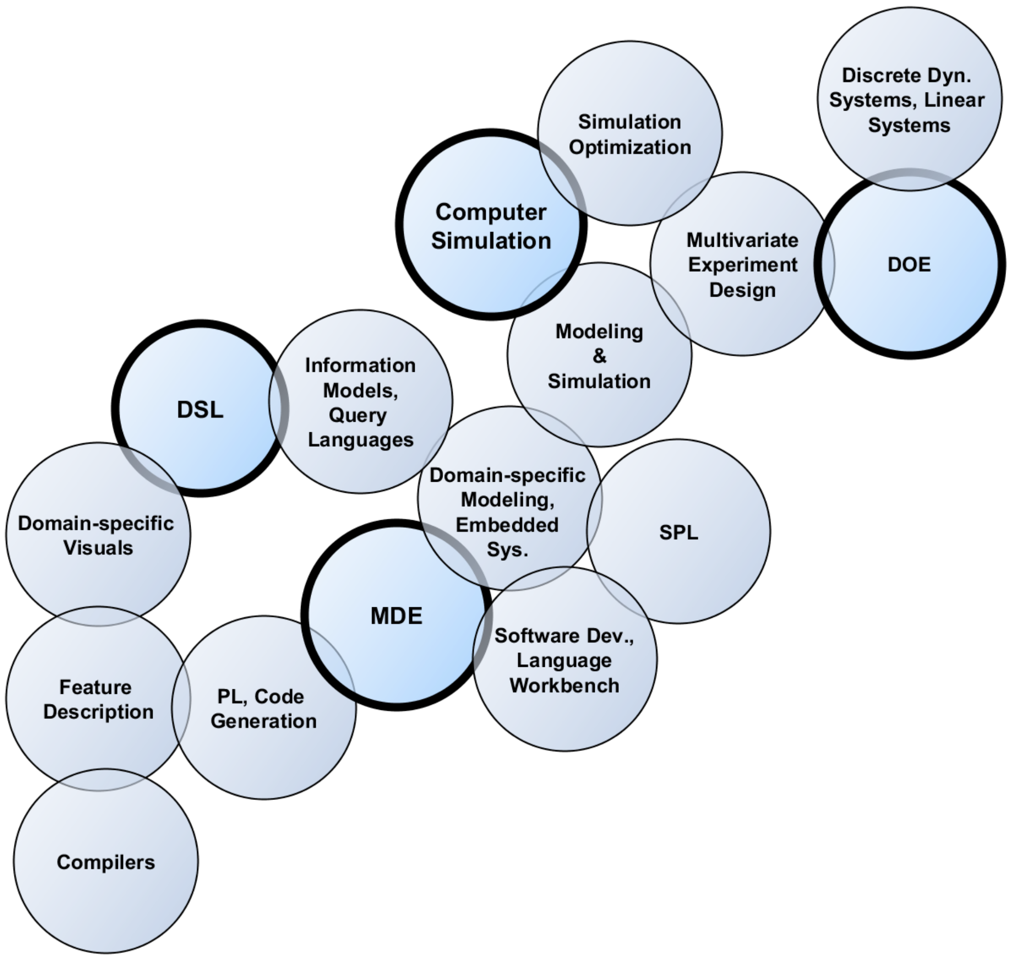

Discrete Dyn. Systems, Linear Systems
Simulation Optimization
Computer Simulation
Multivariate Experiment Design
DOE
Modeling & Simulation
DSL
Information Models, Query Languages
Domain-specific Modeling, Embedded Sys.
SPL
Domain-specific Visuals
MDE
Software Dev., Language Workbench
Feature Description
PL, Code Generation
Compilers

Figure 2.1: Knowledge Map (key topics are highlighted)

CHAPTER 3

OVERALL APPROACH

The main purpose of this chapter is to convey the overall approach by discussing the components of the proposed solution. Although the tangible part of the solution is the experiment design environment, it is only result of much bigger information management activities. Domain knowledge and consolidated formal models are used to create these tools and these tools are used to facilitate the practitioners' work.

3.1 Building Blocks

Experimental design enables users to systematically analyze the outcome of the both physical and simulation experiments [33]. Different statistical models exist for analyzing designs, so that a well-designed experiment provides us with a greater confidence in the results. Thus, it is important to enhance replicability of the simulation experiments. The objective of our work is to produce a user-friendly framework for experiment specification and execution. As originally defined in [13], "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 directly a domain-specific language. Figure 3.1 is the high level pictorial representation of our approach.

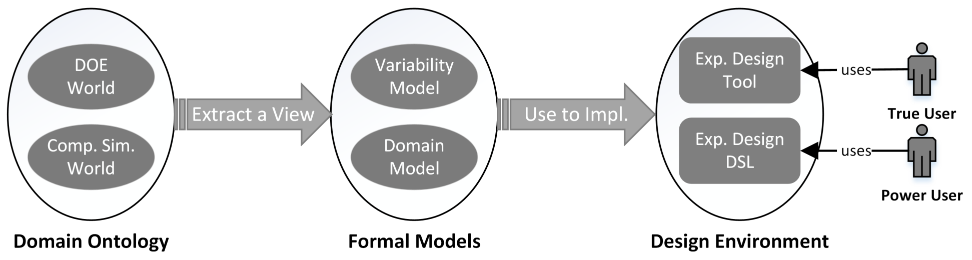


Figure 3.1: Overall Approach

Although the DOE (Design of Experiment) and computational simulation domains are not explicitly formalized, they are the main knowledge sources of our formal models (variability and domain models in Section 3.2). Feature model (variability model) and simulation experiment model (domain model) are constructed using our domain knowledge (refer to Section 3.2 for details). These two models enable Model-to-Text (M2T) transformation at the implementation level. According to our approach experiment replicability is handled by code generation [34]. Code generation supports variability directly, does enables us to cope with variations in operational environment. Therefore, these models are used to implement the experiment design environment and the DSL (grammar and tool-chain). The specification language (DSL) is a critical aspect of the approach since its expressiveness defines the capabilities of the users. The real value of transforming high level specifications into executable experiments is twofold: First, step-wise refinement engenders different level of interactions. For instance power user defines features, experiment components and their relations and then true users refine by selecting experiment features only. Second, the generation of the whole boilerplate code makes polyglot simulation experiments possible (involving multiple programming/simulation languages). The users are not bound to a single execution environment. It is possible to have a simulation model in one language and analysis code in another one by using our approach (e.g. MATLAB simulation model can be run by experimental frame provided by R scripts and/or Java code).

3.2 Formal Models

In the previous section, the solution components are defined. Accordingly, computer simulation and DOE concepts are discussed in the Chapter 1. The main purpose of this section is to discuss other important components of the solution; the formal models (see Figure 3.1)

3.2.1 Variability Model

In order to analyze a particular set of systems according to commonalities, variability models are used to capture characteristics of this set. A primary concern of variability modeling is to expose features in terms of commonalities and variances effectively. The following subsections define the notation briefly and discuss our approach on variability modeling.

3.2.1.1 Feature Modeling

A feature model is a general representation of the possible products of a product family. Feature modeling, and, in general, variability modeling is an active research area since the seminal work of Kang *et al.* on FODA [35]. Feature modeling is an activity to model problem space. Therefore, it involves high level definition of requirements. The model provides an abstraction for the users (true users in our case) to allow them to specify their own requirements just by selecting required features. Feature tree is the most commonly used notation to represent feature models and Figure 3.2 depicts the core elements of this notation.

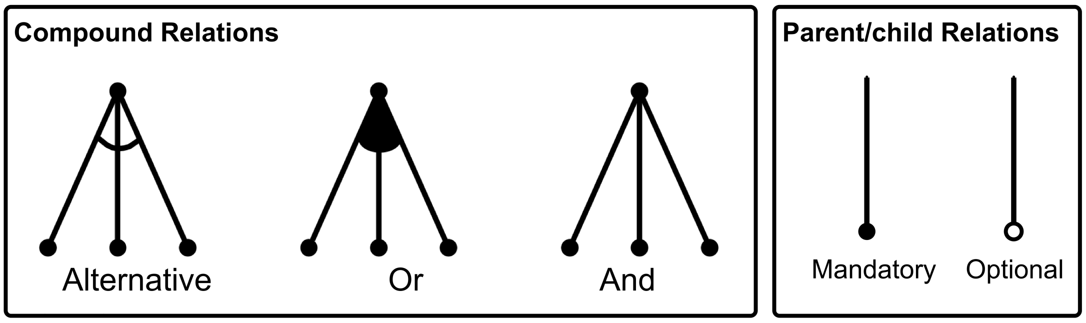


Figure 3.2: Feature Tree Notation

The notation is used to frame problem domain by defining valid configurations. Even a relatively small feature tree may host a lots of different valid configurations (see Figure 3.3). In this respect, formalizing problem space (domain) knowledge by using constraints (cross-tree, compound/parent relations, etc.) is vastly important to create a sustainable model.

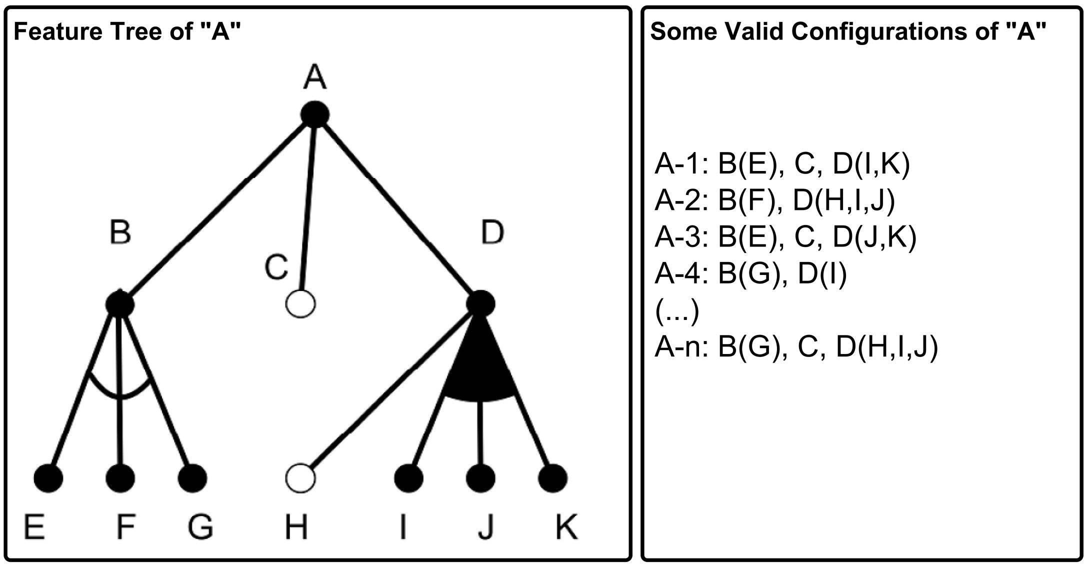


Figure 3.3: Feature Tree and Feature Configurations

3.2.1.2 Simulation Experiment Feature Model

In terms of simulation experiment design [36], features can be categorized into two groups. First, the simulation related features. These features are related to the aspects such as model type, random number generation method. Our environment uses these configurations for the execution of the simulation model. Second, the DOE related features such as objective, a number of factors and analysis method. As the name implies, this configuration group is used to manage the workflow of the experiment. In [33], there is an extensive list of DOE related concepts, but since our scope is limited to simulation experiments, the design techniques like blocking or confounding is not covered (these techniques are applicable to real-world observations only). Although, the current set of features is adequate in many cases, the authors do not claim an exhaustive set. As per our approach, the feature model is planned to evolve iteratively in the course of time. As the feature model gets more refined in time, we expect more components being identified for reuse (see Section 3.2.3 for the details).

- **Simulation Model:** According to the adopted time approach, the model can be dynamic (state variables change over time) or static (snapshot at a point in time). Dynamic models are divided into two classes; continuous (variables change continually with time) and discrete-event (variables change at certain points in time). Using a combination of discrete and continuous variables is also possible (via mixed or hybrid simulation).

- **Pseudo-random Number Generation (PRNG):** Pseudo-random number generators are used to introduce randomness into the model. It is an optional feature because it is only valid on stochastic models and deterministic simulations do not need that feature. User can select one of the different PRNG methods that are alternative to each other (For the sake of simplicity, it's limited to three methods; Linear Congruential, Lagged Fibonacci, Mersenne Twister).

- **Objective:** The goal of the simulation experiment influences the execution and analysis of the experiment. In general, there are three types of objectives of experiments [1]: (i) Comparative designs are related to compare the effects of the factors (Choosing between alternatives). (ii) Screening designs cover monitoring the effects of one or more factors. Lastly, (iii) response surface designs aim to reduce variation of results in a specific value range.

- **Number of Factors:** Since computational resources are often limited, the number of factors is a vital feature in designing efficient experiments. For instance, some sampling methods are not practical for more than four factors (at least using commodity hardware).

- **Sampling Method:** In terms of DOE, result instances are the outcome of iterations (running the model) and each iteration is related to a specific design point (determined by the values of all factors). In this respect, sampling method is a method that specifies the values of factors for all design points. Table 3.1 depicts adopted constraints for the variability model on objectives, a number of factors and sampling methods.

Table 3.1: Design Selection Guidelines (adapted from [1])

Num. of Factors	**Comparative Objective**	**Screening Objective**	**Response Surface Obj.**
1	Randomized design	N/A	N/A
2-4	Randomized block design	Full or fractional factorial	Central composite or Box-Behnken
5 or more	Randomized block design	Frac.factorial/Plackett-Burman	Screen first to reduce num. of factors

- **Analysis Method:** De facto analysis method of DOE is analysis of variance (ANOVA). Nevertheless, design in use influences the application of that method. For example, if there exists more than one response variable, MANOVA (Multivariate Analysis of Variance) is used. T-test and F-test of ANOVA are equivalent (One-way ANOVA) if there exists only one factor (which means two groups/levels to compare). Thus, one factor case can be covered by T-test, and Factorial ANOVA is used to study more than one factor with a single response variable.

The feature model in Figure 3.4 is a formal representation of the above definitions and their interdependencies. In this model, abstract features are used to support understanding; these features are not mapped to any implementation artifact [37]. Other features (Concrete ones) on the other hand, have some mappings to experiment implementation. For instance, `Analysis Method` is mapped to implementation of typical DOE plots (main effects mean, interaction plots, etc.) while the function of the `Objective` feature is simply grouping its sub-features.

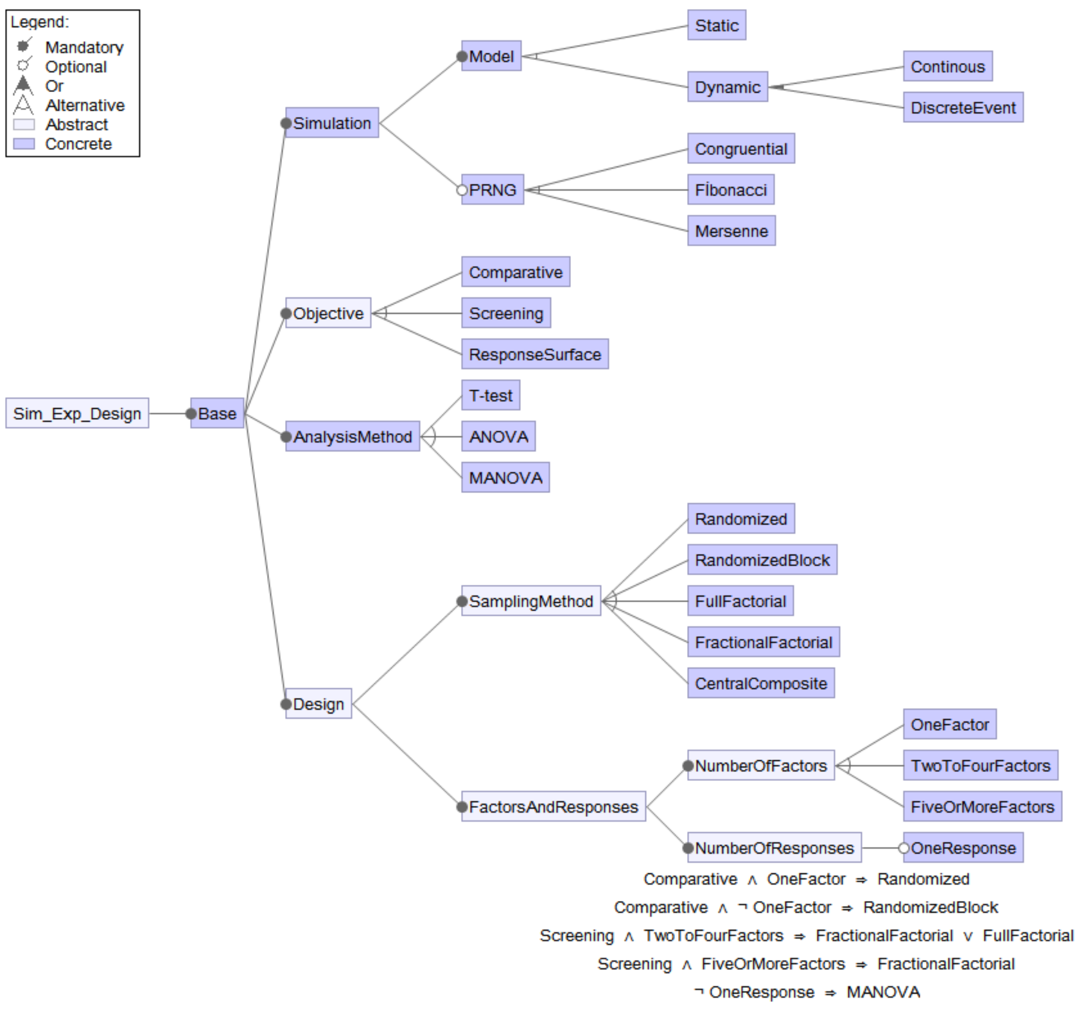


Figure 3.4: Feature Model of Simulation Experiment Design

3.2.2 Domain Model (Simulation Experiment Model)

The purpose of this section is to present and describe the proposed simulation experiment model. It provides a conceptual understanding of the components of a simulation experiment as well as the relationships between them. Furthermore, it provides a domain model on which simulation experiments can be based (see Figure 3.5). The model is also used to shape the DSL grammar.

- `Experiment`: The root element is the Experiment class. The attributes of this class include relevant information about the experiment, such as the experiment name, date and description. Additional information about the experiment can also be included, such as the experiment cost, the experimenter's name, and so on. As can be seen in the model, an experiment is composed of following main components: Simulation model, objective, simulation runs, design, design matrix, statistical analysis and visualization methods. A detailed description of these follows.

- `SimulationModel`: It is the core part of the simulation experiment. In conventional experiments, the very first step of the overall process is to identify the experimental unit (person/object that will be studied) that provides sampling data. In simulation experiments, the simulation model is the primary source of information.

- `Objective`: Defines what the purpose of the experiment is. This definition influences the experiment type and the number of runs that are required to achieve the experiment's goals. In this early form of model, objectives are defined as a free form text, but it's an ongoing task to formalize the objectives (by constructing relations with the other components of model, such as `Variable`).

- `Run`: The number of experiment runs required is not necessarily known beforehand; it may depend on the actual progress of the experiment. Each run has a start and end time.

- `Design` and `DesignMatrix`: Experiment design depicts the structural aspect of the experiment. The experimental structure is defined by the responses, the factors and their levels and user provided value ranges. Based on the design, a design matrix can be created, which specifies the actual experimental runs, that is, the combination of factor levels to be tested.

  The design matrix is the actual implementation of the design. Each row of the matrix corresponds to a factor level combination and the cells for the responses will be filled in as the experiment is executed. In the case of a simulation experiment, the factor values of each factor-level combination are the values of the input parameters that are being tested.

- `StatAnalysis`: The response values and even the list of significant factors by themselves are useful, but without further analysis, are limited in the value they can provide to an experimenter. Therefore, statistical analysis methods (such as ANOVA, MANOVA, ANCOVA, MANCOVA) on this data can provide a wealth of useful information and knowledge about the influence of factors on the responses. There are many different kinds of statistical analyses available, but we limit ourselves to the most common ones.

- `Visualization`: Presenting data in tables may not be the best way of communicating the results of an experiment. Graphs and charts are valuable tools that can convey the information more effectively. This part of the model denotes the method of visual representation of the analysis.

- `Variable` (`Response` and `Factor`): Each variable is identified by its name and type. Currently, four types of variables (Integer, Float, Boolean, String) are supported in the model. These variables can be divided into two distinct classes; responses and factors. Responses represent the output values of the experiment. Responses correspond to the dependent variables. We wish to analyze and characterize the effect of the factors on these responses and, hence, must record their values at varying factor level combinations. Factors correspond to the independent variables, control variables and nuisance variables. These factor types are distinguished by the attribute `group`.

  An experiment is conducted by varying the factor values and recording the outcomes. Each factor can have multiple levels (treatments). There is a one-to-one mapping between a factor level and a factor value. The factor values are the values that are actually fed into the simulation model. Response values are generated at each experimental run. An experimental run is a run of the simulation experiment with a set of input parameters. The input parameters are the factor values for that run. The combination of these values is known as a factor-level combination. They correspond to a row of the design matrix. The response values are also recorded in the design matrix.

- `SamplingInstance`: Sampling instance is basically an aggregation of a variable and its actual value. It can be an input to the model (factor variable) or output of it (response variable). Sampling instances appear in a design matrix.

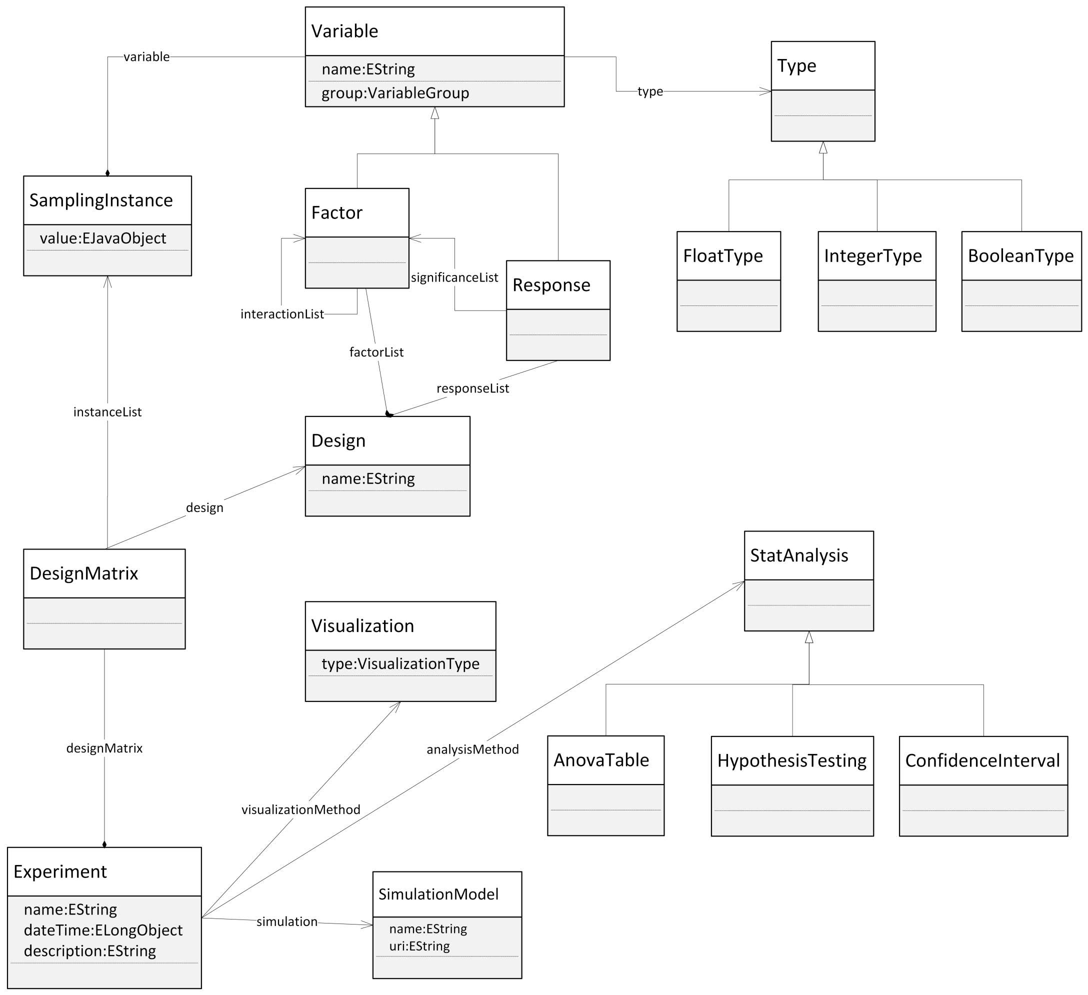


Figure 3.5: Simulation Experiment Domain Model (simplified)

3.2.3 Variability Management

In exploring phenomena, the scientific method can be construed as a process with multiple stages. The scientist poses a hypothesis and deduces its predictions, which drive an experiment (you can refer to Chapter 1 for the details of our interpretation of the scientific method). Nevertheless, it is not always possible to draw a reliable conclusion with a single prediction, especially for non-trivial cases like the ones in computational biology [38]. Such difficulties bring an intuitive loop to the method. Additional predictions are made due to the results supporting the hypothesis or the hypothesis is revised due to a result that does not support its predictions. Therefore, the general use case of our framework covers a family of experiments, rather than a single one. To meet that demands, we have to deliver experiment variations while keeping the family assets coherent. This is the main motivation to find a formal way to manage variabilities among a set of experiments. Our solution to this design problem is to adopt variability management with feature modeling approach [39]. By using a formal approach like feature modeling brings out-of-the-box solutions to many problems (use cases). For instance experiment iterations may require unforeseen variations and it can be handled by extending feature models and map these features to experiment assets.

3.2.4 Management Process

The possible choices regarding the requirements are formalized in the variability model (Section 3.2) which is an abstraction of the variability in the simulation experiment domain requirements. The Chapter 4 explains the details of our language-based approach to manage experiments. The language allows for the specification of single or a family of experiments. Furthermore, the language is extended to annotate the artifacts with features (for the details of similar superimposition method, please refer to [40]).

The basic idea is quite simple:

1. The user (possibly a power user) prepares experiment family definition and annotations.
2. The user (possibly a true user) selects the required features to specify their actual experiment.
3. Then, the specified feature configuration is used to include/exclude annotated parts of the low level experiment definition (written in our DSL by a power user).

This workflow is illustrated in Figure 3.6. True users configure the feature model to specify actual experiments while power users define experiments and their variations (experiment family) by using our DSL.

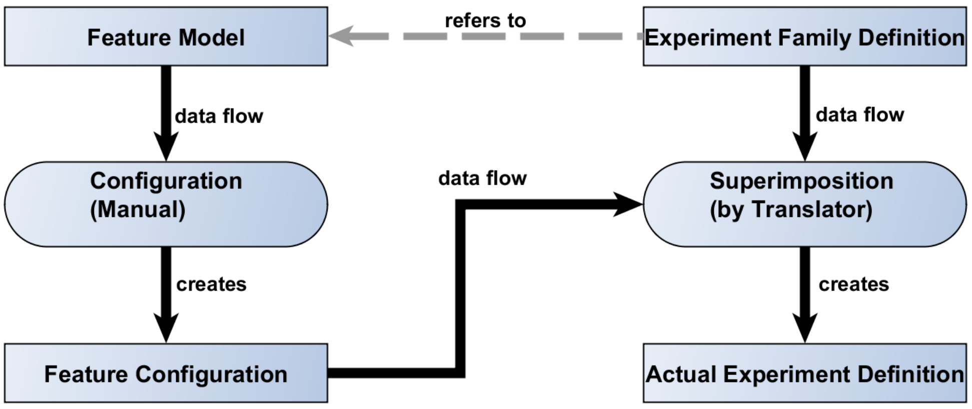


Figure 3.6: Variability Management Approach

3.2.5 Applying Configurations

The major objective of this study is to find an effective way to manage a family of simulation experiments. In this respect, using feature models to manage variability pays off when users equally or more prefer to use it to execute experiments. In this respect, application of the process is another important aspect.

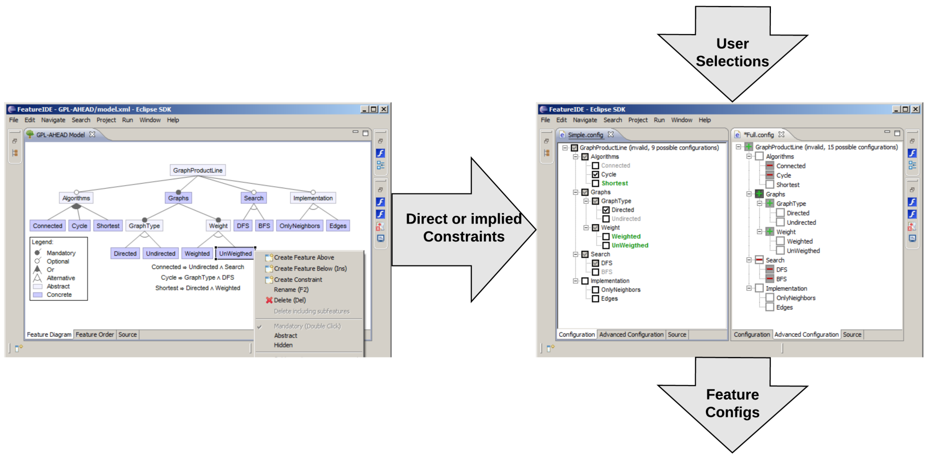


Figure 3.7: Creating a Feature Configuration with the FeatureIDE

Feature trees consist of nodes and relations among them (parent/child, inter-tree, constraints, etc.). As we discussed earlier, according to our solution approach the consolidated knowledge in the form of feature tree is used to manage the configuration phase. Although there exists a lot of alternative tools/environments, we adopted to use FeatureIDE [41] for this task. Our code is able get the feature configurations (a text file including comma-separated feature names) created by FeatureIDE as far as annotations and feature names in the tree are matched. It is assumed that all consistency/validity checks are handled by the tool (FeatureIDE checks these kind of constraints and outputs valid configurations only). It is relatively easy task to switch that part for the use of another tool since it is very straightforward (just matching feature names). In Figure 3.7, feature tree (at the left) and feature configuration (at the right) editors are shown with their information flow. This flow depicts how "Manual" configuration is handled by user in Figure 3.6.

CHAPTER 4

DOMAIN-SPECIFIC LANGUAGE FOR SIMULATION EXPERIMENT DESIGN

The main focus of this chapter is take a closer look to the details of the introduced DSL. Different functions such as using the DSL to specify an experiment or variants of an experiment family are discussed in individual sections.

The objective of introducing our own language is twofold: First, it is the medium of specifying simulation experiments. Users are able to declare their requirements in the problem space and these declarations are used to generate respective solution artifacts by the language translator. Secondly, fragments of an experiment specification are mapped to features (see Section 3.2.3) to manage simulation experiment variability.

EMP - Eclipse Modeling Project [42] is used to implement the domain-specific language and its environment. We mainly focused on designing the grammar since the EMP platform handles the heavy lifting of language engineering tasks. The grammar is a vital part of the language. Its content and form decide the level of usability. The ease of use is one of our primary concerns and we tried to contribute that feature by designing an elegant grammar. Although the following subsections give the details of different aspects of the language with the examples that give a hint on the backbone of the grammar the reader can refer to the dedicated project website (https://odayibas.github.io/xperimenter/) for the complete definition.

4.1 Specifying an Experiment

The `Experiment` has some properties that are identified by the domain model (see Figure 3.5) and some of them (e.g. `timeout` and `description`) are optional to support ease of use. In addition, some properties (e.g. `Design`, `Simulation`, `StatAnalysis` references) are inherited from the feature model (see Figure 3.4). Although these references are deducible by using feature configuration, they are also defined in the grammar for the sake of creating a self contained DSL. The experiment definition is not the only source of information to the translator (refer to Section 5). True users are advised to use feature configuration files to create variations of the base experiment.

```
experiment exp_A
{
  desc "brief description of the purpose of the
     experiment";
  objective COMPARATIVE;
  design FULL_FACT_DESIGN;
  simulation SIMULATION_A;
  analysis STAT_MTD_ANOVA;
  visual BARCHART;
  timeout 180;
}
```

The `experiment` definition involves a reference to an experiment design and each `design` is given a name (to refer to it), a sampling method to be used, and a list of variables:

```
design FULL_FACT_DESIGN
{
  method FULLFACTORIAL;
  varlist X,Y,Z;
}
```

The DSL assumes that the `modelFile` is a runnable entity in the target platform (Kepler is the only supported platform right now) in order to provide a platform independent way to execute the simulation model. Additional information like `modelType` is left optional because current implementation does not use it, but it might be used for prospective target platforms (see Chapter 5 for details). A similar approach is also used to define the statistical analysis method.

```
simulation SIMULATION_A
{
  modelFile tr.edu.metu.ceng.xperimenter.model.SimA;
  modelType CONTINUOUS;
  inport inX : X;
  inport inY : Y;
  inport inZ : Z;
  outport outZ : K;
}

statAnalysis STAT_MTD_ANOVA
{
  service tr.edu.metu.ceng.xperimenter.statistics.Anova;
}
```

Aforementioned `design` entity involves a set of variables (in fact, references to these 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:

```
variable X : FLOAT group FACTOR [0.5, 1.5];
variable Y : INTEGER group FACTOR [0.3, 2.4];
variable Z : FLOAT group NUISANCE gen NUM_GEN_1;
variable K : FLOAT group RESPONSE;

generator NUM_GEN_1
{
  service tr.edu.metu.ceng.xperimenter.numgen.Mersenne;
  seed 1024;
}
```

Although the above examples give a hint on the backbone of the grammar the reader can refer to the project website (https://odayibas.github.io/xperimenter/) for the complete definition.

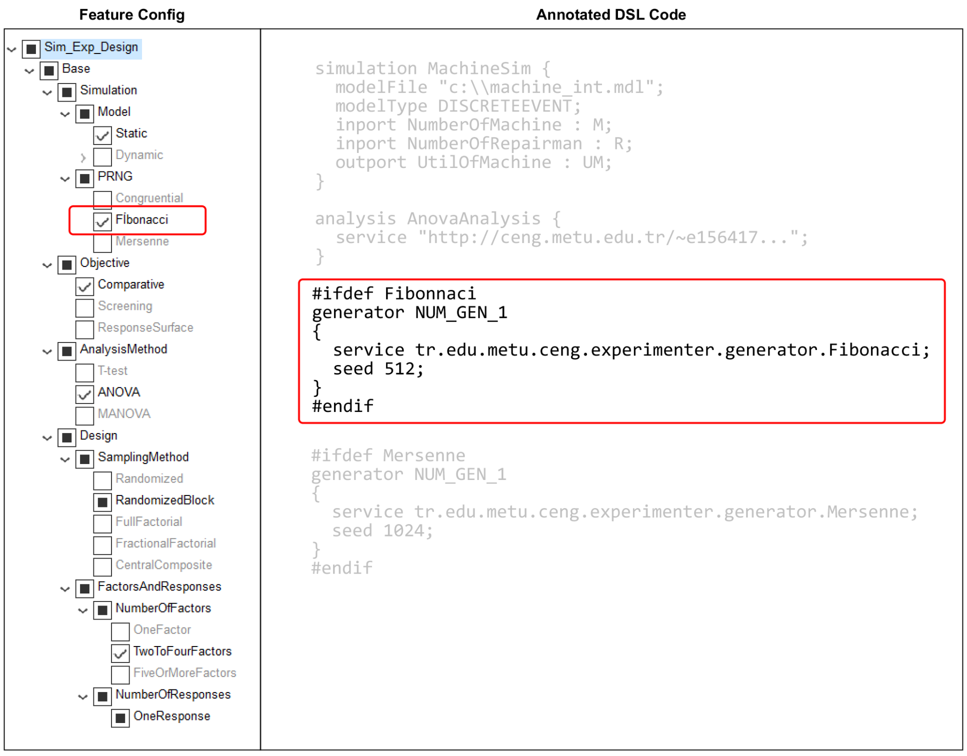
4.2 Specifying Variants of an Experiment Family

In terms of variability management, an experiment family defines a set of variants that can be selected by the user to specify a member of that family (a particular experiment). A way to annotate DSL snippets with features need to be provided in the DSL environment. In section 3.2.3, the variability management approach is introduced. The design environment applies that method by using preprocessor directives. It is very similar to C-style conditional compilation.

```
#ifdef Mersenne
generator NUM_GEN_1
{
  service tr.edu.metu.ceng.xperimenter.numgen.Mersenne;
  seed 1024;
}
#endif

#ifdef Fibonacci
generator NUM_GEN_1
{
  service tr.edu.metu.ceng.xperimenter.numgen.Fibonacci;
  seed 512;
}
#endif
```

The above method allows a user to specify a set of variants and to map them to the feature model. The resulting specification creates a feature annotated code (it is also another feature model) and then the language processor superimposes the selected feature configuration onto the provided code to extract a particular valid experiment specification. For instance, above code snippet includes two random number generators with the same name. According to the DSL grammar, these two generators cannot reside in a single experiment (see Figure 4.1).



Feature Config

```
■ Sim_Exp_Design
  ■ Base
    ■ Simulation
      ■ Model
        ✓ Static
        ☐ Dynamic
      ■ PRNG
        ☐ Congruential
        ✓ Fibonacci
        ☐ Mersenne
    ■ Objective
      ✓ Comparative
      ☐ Screening
      ☐ ResponseSurface
    ■ AnalysisMethod
      ☐ T-test
      ✓ ANOVA
      ☐ MANOVA
    ■ Design
      ■ SamplingMethod
        ☐ Randomized
        ■ RandomizedBlock
        ☐ FullFactorial
        ☐ FractionalFactorial
        ☐ CentralComposite
      ■ FactorsAndResponses
        ■ NumberOfFactors
          ☐ OneFactor
          ✓ TwoToFourFactors
          ☐ FiveOrMoreFactors
        ■ NumberOfResponses
          ■ OneResponse
```

Annotated DSL Code

```
simulation MachineSim {
  modelFile "c:\\machine_int.mdl";
  modelType DISCRETEEVENT;
  inport NumberOfMachine : M;
  inport NumberOfRepairman : R;
  outport UtilOfMachine : UM;
}

analysis AnovaAnalysis {
  service "http://ceng.metu.edu.tr/~e156417...";
}

#ifdef Fibonnaci
generator NUM_GEN_1
{
  service tr.edu.metu.ceng.experimenter.generator.Fibonacci;
  seed 512;
}
#endif

#ifdef Mersenne
generator NUM_GEN_1
{
  service tr.edu.metu.ceng.experimenter.generator.Mersenne;
  seed 1024;
}
#endif
```

Figure 4.1: Feature configurations and annotation use

The simulation experiment feature model allows the selection of only one of the alternative options in a valid feature configuration (see the “alternative” relation in Figure 3.4). Feature configuration environment (e.g. FeatureIDE) enforce direct or implied constraints during the selection operation. Therefore, the actual experiment may include only one generator definition according to user provided feature configuration. As per their primary use case, true users use wizard-like selection tool (see the check boxes to select concerning features in Figure 4.1) to manage experiment variations. Similarly, the base feature (always selected root feature) denotes the commonalities of the experiment family (shared code snippets like the core assets of the experiment).

CHAPTER 5

GENERATING AN EXECUTABLE EXPERIMENT

There is a constant trade-off between increasing level of abstraction to facilitate the management process and decreasing for operational purposes. As per our approach, model transformations are key methods to change levels of abstraction when it is needed. These transformations are used to generate platform-specific assets such as executable experiments. This chapter is dedicated to elaborate how an executable experiment is generated in our proposed framework.

5.1 Power Users vs. True Users

As we already covered in the previous sections, our framework is designed to serve two distinct user groups; true and power users (please refer to Chapter 3 for the details of this classification). In this respect, experiment execution use case of these two groups differs from each other. As a preprocess we assume that the experiment family specification is curated and required annotation is also defined (see Section 3.2.3). According to our approach, this process is power user's responsibility. Figure 5.1 depicts the sequence diagram of a power user.

In this scenario, power user uses his/her coding skills to specify the experiment. First, `Editor` checks the code and updates the user interface to highlight possible inconsistency. It passes the specification to the `Interpreter` and it generates corresponding artifacts. At this stage, we have platform-specific artifacts and these artifacts are executable in only one environment. Thus, submitting them to the concerning environment handler such as `KeplerHandler` is the final step of the process.

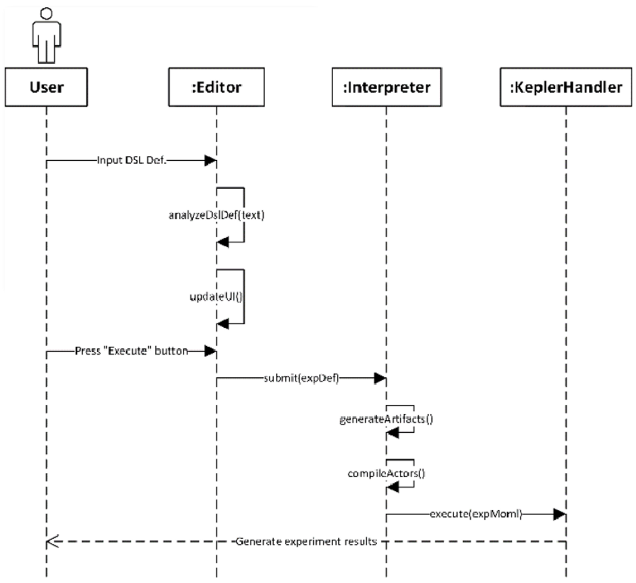


Figure 5.1: Experiment Execution Sequence Diagram for Power User

True user on the other hand, uses the feature model configuration tool (it is a kind of wizard to make the process easier while sacrificing comprehensiveness) as a primary tool. In this case, feature configuration specified by the user is used to generate corresponding experiment definition. As shown in Figure 5.2, the rest of the process is similar to the sequence of the power user.

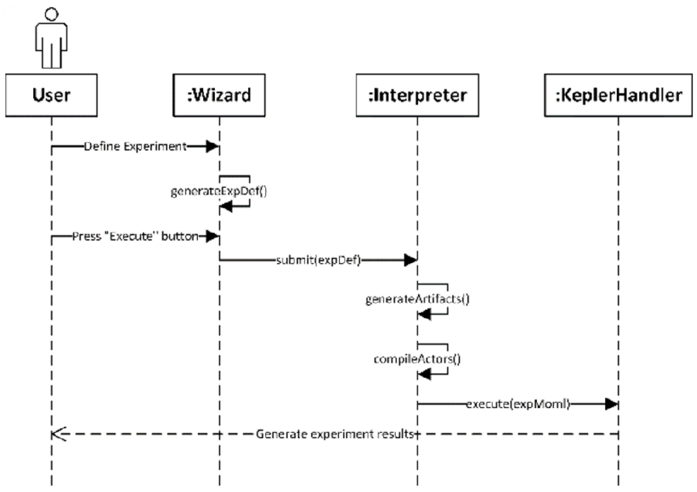


Figure 5.2: Experiment Execution Sequence Diagram for True User

5.2 Target Environments

Although our approach is platform agnostic, the Kepler [10] environment is selected for demonstration purposes in the present implementation. Kepler is an environment for the design and execution of scientific workflows. It is based on the Ptolemy II system which relies on actor-oriented modeling paradigm [43]. In this paradigm, two main entities constitute the backbone of the workflow. First, the "Director" manages the data flow, and second, a set of "Actor"s executes the steps of the workflow. Although the experiment workflow is static [44], the "Actor"s in action are subject to change according to the feature configuration and DSL statements.

In Section 3.2.2, the artifacts that constitute the runnable experiment are listed in detail. In terms of Kepler, these artifacts need to be represented by respective actors. Figure 5.3 depicts our transformation approach to generate workflow components. The translator uses the definitions provided by the user(s) and constructs all artifacts as a scientific workflow. This final output is ready to be run on the Kepler environment.

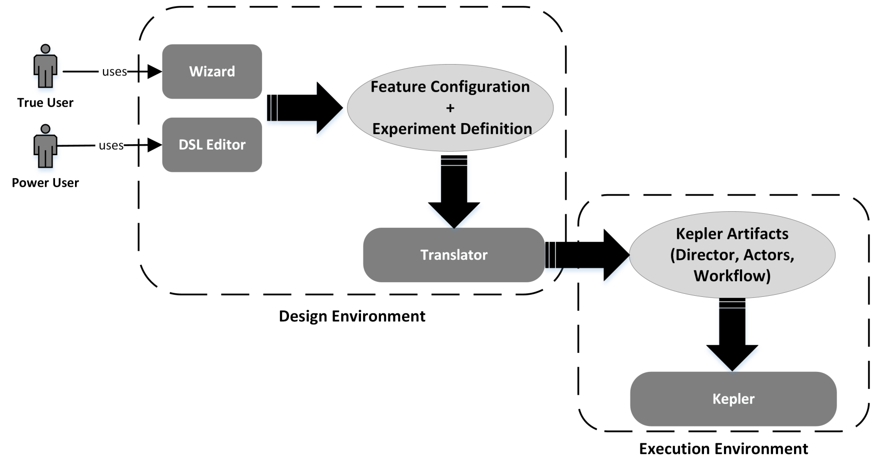


Figure 5.3: Conceptual View of Our Generative Approach

Kepler inherits the actor-oriented paradigm of Ptolemy II, and the use of actors is the main method to provide separation of concerns and reuse. Resulting experiment workflow employs a director and a set of actors. `DesignMatrixManager`, for example, is the actor that generates the sampling instances (design points). The other function of that actor is to accumulate the results of the individual runs. The duty of `Terminator` is to decide whether additional runs will be executed or not due to the external constraints such as a time limit. As the name suggests, `ModelRunner` runs the simulation model by using the provided parameters. Furthermore, `StatAnalyzer` uses the statistical analysis method on the filled design matrix. In Kepler, DE (discrete event) actors fire only after they receive their inputs. Thus, `SampleDelay` is a required actor due to the nature of DE director. It prevents the workflow from falling into a deadlock.

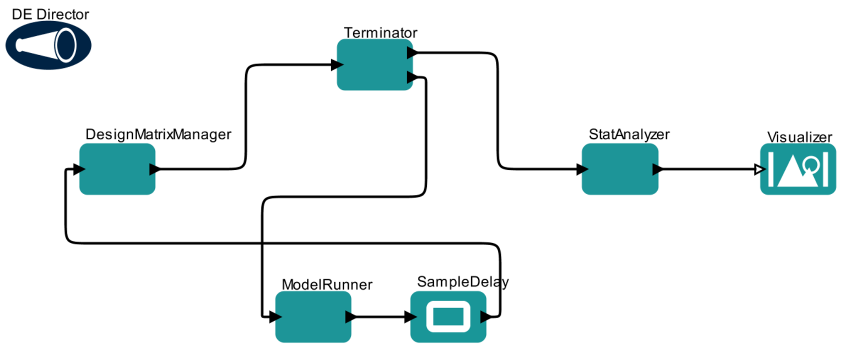


Figure 5.4: Generated Kepler Workflow (arrows denote data precedences)

CHAPTER 6

CASE STUDIES

Building blocks of our proposed approach have been covered in the previous chapters and this chapter is solely reserved to demonstrate the approach by giving use cases from diverse application areas. Each section individually defines the problem and use of our approach to handle situation.Respectively, "machine interference" focuses on generating an executable simulation experiment while "quadcopter controller" one is more related to demonstration of variability management of the framework and final "airline flight revenue model" exemplify the use of a stochastic model.

6.1 Machine Interference Experiment

6.1.1 Problem Definition

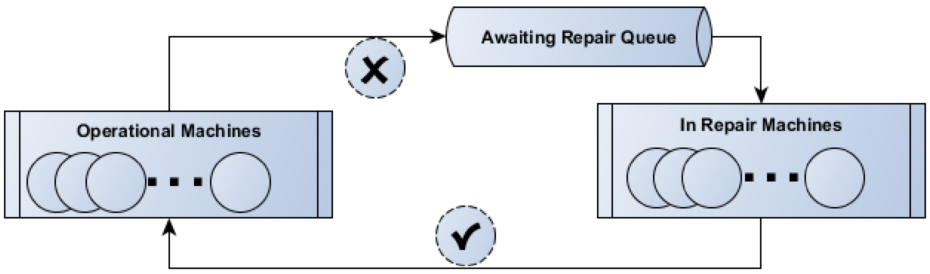


Figure 6.1: Machine Interference Problem

Let us consider the machine interference problem [45] to demonstrate our approach. There are M machines and R repairmen. Each machine is operational for a period of time until it needs maintenance. The repairmen serve the machines in a FIFO manner and the repair operation is non-preemptive (see Figure 6.1). The time required to fix a machine is not constant; it is a random amount of time. Assume that we have a discrete-event simulation model for this problem and that model has the following external variables.

- External inputs:
  - **M:** number of machines
  - **R:** number of repairmen
- External outputs:
  - **UM:** utilization of the machines (total operational time / total time)
  - **UR:** utilization of the repairmen (repair time / total time)

6.1.2 Experiment Specification

At this stage, we can define our experiment informally. Assume that our research question is "Which is the best way to increase UM; buying a new machine or hiring a new repairman?". This is a comparative experiment where M and R are factors and UM is a response variable. The DSL code below encodes one of the possible alternative experiment definitions to conduct this experiment.

```
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, 5];
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";
}
```

The process of experimentation for the machine interference case study is illustrated by the activity diagram in Figure 6.2. As a first step, the translator decides whether the definition is for a single experiment or a family of experiments. If there exist unresolved variations, meaning that it is for a family of experiments, it resolves them according to user-provided feature configuration. In this example, the experiment does not have any variability. Thus, it can be transformed into the execution environment without pre-processing. Translator creates a `DesignMatrixManager` that builds a design matrix according to the specified design (full factorial) and prescribed high/low values of the factors. The other generated actor `ModelRunner` includes codes to run the simulation model. Basically, it executes the `machine_int.mdl` (a Matlab/Simulink model) file by setting the actual values of its external variables, namely, `NumberOfMachines` and `NumberOfRepairmen`. In a non-interrupted experiment, the model is executed for each row of the design matrix; however, an explicit time limit (180 seconds) is specified in our example. Thus, after each run, `Terminator` checks whether that limit is exceeded or not. After the model execution loop is terminated, `StatAnalyzer` applies ANOVA to the filled design matrix and generates analysis results. In our case the user requested standard ANOVA analysis with its default visualization. Therefore the final output of the workflow includes the resulting ANOVA table and the multiple comparison chart. As a side product, the design matrix is also generated in CSV (comma separated value) file format.

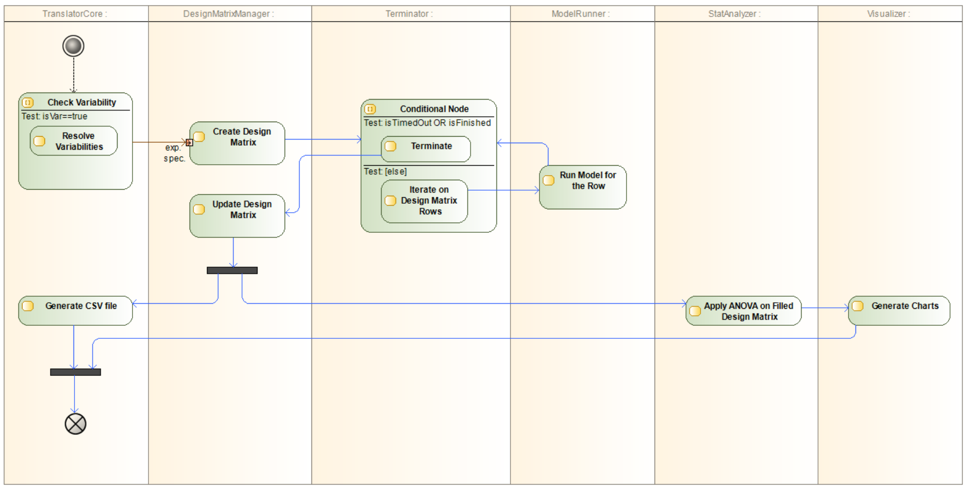


Figure 6.2: Activity Diagram

6.2 Quadcopter Controller Experiment

6.2.1 Problem Definition

Inherently, a multi-rotor systems like quadcopters have coupled rotational and translational subsystems. It is not possible to translate the system to anywhere without any rotational force. Therefore, overall system dynamics are nonlinear and complex in these systems. Even a human control (e.g. remote controller) is not be feasible without electronic stabilization assistance system in quadcopters due to the this complexity. That is to say, the performance of the electronic controllers are indispensable part of the whole user experience.

As a controller, PD controllers are easy to apply because of theirs simplicity but they are also inadequate for many use cases. PID controllers creates smaller steady-state error but they also have some shortcomings like reset windup (integral saturation). Therefore, finding right controller configurations (Tuning) is an important part of designing a real world solution by using a quadcopter platform. The goal of this case study is demonstrating our proposed experimentation framework to handle this task.

6.2.2 Experiment Specification

As the name suggests, Proportional-Integral-Derivative (PID) control consists of three basic coefficients; proportional, integral and derivative which are varied to get optimal response. Figure 6.3 depicts the general principles and execution of this type of controllers.

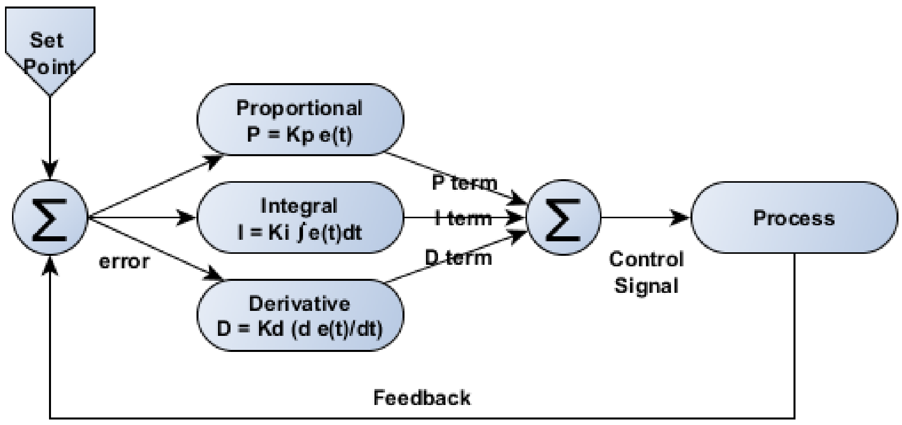


Figure 6.3: PID Controller

The proportional component depends only on the difference between the set point and the process variable. This difference is referred to as the error term. The proportional gain (Kp) determines the ratio of output response to the error signal. The integral component sums the error term over time. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless the error is zero, so the effect is to drive the steady state error to zero. The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable. Ideally, we would be able to use a method to analyze a system and output the "optimal" PID gains (Kp, Ki, Kd).

As we already discussed, the quality of a controller depends on the gain values. On the other hand, tuning these parameters require expert intuition and a lot of time. On the other hand, using our framework helps to decrease the total effort. Step by step instructions listed below.

1. Define a research question. Our goal is to find an answer to the following question.
    - "Which gain parameter is more important to determine the quality of the controller?"
2. Prepare the required models to interface with external entities.
    - We are using the third party quadcopter flight model which is publicly available on Github as a MATLAB code [46].
3. Write the experiment definition in the Xperimenter.
    - Below snippet depicts a full factorial design of this experiment in our DSL.

```
experiment QuadcopterExperiment {
   desc "Which PID parameter effects the
      controller quality most"
   objective COMPARATIVE;
   design CompQuadcopterExpDesign;
   simulation QuadcopterSim;
   analysis AnovaAnalysis;
   visual DEFAULT;
}
```

4. Get the resulting ANOVA table and evaluate it.

- The response table (which has a thousand of rows) is the input of the analysis. Below you can find the the resulting table.

Table 6.1: Quadcopter controller experiment ANOVA Table

Source of Variation	**Sum of Squares**	**Degrees of Freedom**	**Mean Squares**
Kd	0.0006	7	0.00008
Kp	0	0	0
Ki	0	0	0
Kd*Kp	0.0079	68	0.00012
Kd*Ki	0.0035	66	0.00005
Kp*Ki	1.5675	46	0.03408
Kd*Kp*Ki	15.6592	692	0.02263
Error	0	0	0
Total	28.1525	962	

- In this experiment, we tried to find out which gain parameter is more important to determine the performance of a PID controller. The system under test is a quadcopter simulation (in Matlab) and the cost function (response variable) is angular displacement to reach steady state (angular motion). We have designed a full factorial design and each one of the three gain parameters may have an integer value ranging from 1 to 10. These design points generate a 1000-row response table and multi-way ANOVA analysis is applied to this table. According to the resulting table, group means are not significantly different from the others. Thus, three gain parameters are equally important.

6.3 Airline Flight Revenue Experiment

6.3.1 Problem Definition

Risk modeling and simulations are used by many organizations to optimize their process and take foreseeable actions accordingly. Airline flight revenue models are customized form of risk models. Airline companies use these models to manage ticket prices and maximize the revenue stream.

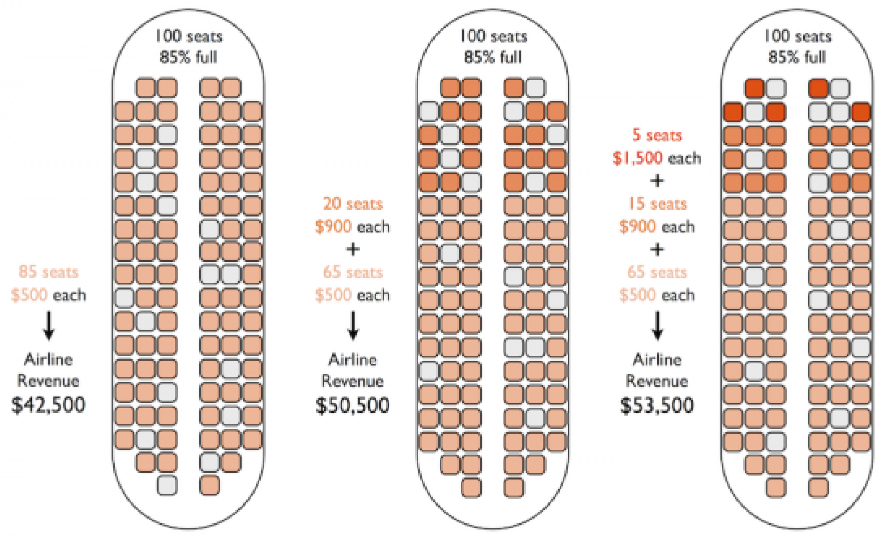


Figure 6.4: Airline Ticket Pricing [2]

Due to practical constraints, this dissertation cannot provide a comprehensive review of risk modeling nor ticket revenue simulations. On the other hand this use case is a good fit to demonstrate the use stochastic simulations and our experiment management approach. Figure 6.4 depicts one strategy on ticket pricing with up-sell. One can create similar or better revenue by selling fewer number of seats. As the example figures proves, best effort selling (try to sell all available seats) is not the best strategy always. However, overbooking (selling more than available seats to compensate no-show customers) may also feasible strategy in many cases.

6.3.2 Experiment Specification

Assume our hypothetical airline company use a simple stochastic simulation to put their strategy of ticket pricing/selling (In real life, the companies use fairly complex models to handle that). The external parameters and their descriptions are as follows:

- External inputs:

- **price:** The price of one seat (Nuisance).
- **capacity:** The number of available seats per flight (Nuisance).
- **no-show:** The number of no-show customers (Factor).
- **compensation:** Compensation payment which is defined by percentage of the ticket price (Factor).
- **refund:** No-show refund payment which is defined by percentage of the ticker price (Factor).
- **sold-ticket:** The number of ticket sold (Factor).

• External outputs:

- **total-rev:** Total revenue of the flight.

Contrary to the previous comparative experiment designs, this one utilizes Response Surface (RSM) to optimize a value. Below you can find the main part of the experiment specification.

```
experiment AirlineTicketRevenue {
  desc   An experiment to find optimum level of
     overbooking to maximize ticket revenue  ;
  objective RESPSURFACE;
  design RespSurfaceAirlineExpDesign;
  simulation TicketOverbookSim;
  analysis AnovaAnalysis;
  visual DEFAULT;
  target R;
}

variable price : INTEGER group NUISANCE 200;
variable capacity : INTEGER group NUISANCE 100;
variable refund: INTEGER group NUISANCE 50;
variable compensation: INTEGER group NUISANCE 125;
variable no_show : INTEGER group NUISANCE gen NUM_GEN_NO
   _SHOW;
```

```
variable sold_ticket: INTEGER group FACTOR [105,125];
variable total_rev: INTEGER group RESPONSE;

design RespSurfaceAirlineExpDesign {
  method CENTRAL_COMP;
  varlist price capacity refund compensation no_show;
}

simulation TicketOverbookSim {
  modelFile "c :  r_models  airline_ticket.r";
  modelType STATIC;
  inport Price : price;
  inport Capacity : capacity;
  inport Refund : refund;
  inport Compensation : compensation;
  inport No_Show : no_show;
  inport Sold_Ticket : sold_ticket;
  outport Total_Rev : total_rev;
}

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

#ifdef Mersenne

generator NUM_GEN_NO_SHOW
{
    service tr.edu.metu.ceng.xperimenter.numgen.Mersenne
       ;
    range [0,24];
    seed 1024;
}

#endif

#ifdef Congruential

generator NUM_GEN_NO_SHOW
{
    service tr.edu.metu.ceng.xperimenter.numgen.LCG;
    range [0,24];
    seed 1024;
}

#endif
```

In this specific scenario, true user compares the alternative random number generator options (Mersenne Twister and Congruential) which is prepared by power user. Figure 6.5 shows this comparison.

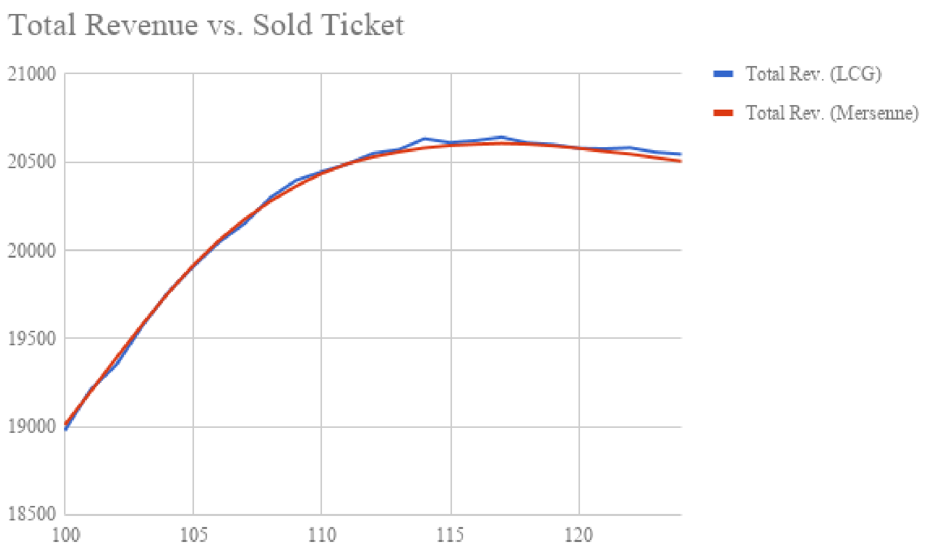


Figure 6.5: Total Revenue Optimization and PRNG Alternatives

The chart in Figure 6.5 proves that around the 17% overbooking maximizes the total revenue and this result is consistent among the use of different random number generators.

CHAPTER 7

CONCLUSION

In this study, a generative approach is introduced to manage the experimentation process with computer simulations. In our approach, domain and feature models are used to capture the relevant aspects of simulation experiments. That information is the main source to manage experiment life cycle operations such as generating corresponding operation instances in execution environments. Our novel contribution is twofold: First, we have defined a formal way to specify and execute an experiment that explicitly supports replicability since it is based on semi-automated model transformations. Second, custom use cases for different levels of users are introduced. To our knowledge, this is the first work to merge the variability management notion with the design of experiment paradigm. Our approach clearly supports systematic reuse by defining a formal way to reuse experiment assets by different levels of users. From the research that has been carried out, it is possible to conclude that model-driven engineering practices have great potential for simulation experiment applications. First of all, true users are completely isolated from glue codes but power users has enough flexibility to define a variety of experiments. Moreover, they both are free to deal with maintenance operations (e.g. platform/formalism changes to adopt a third party experiment) since the translator handles such details. Our framework development is an ongoing work as an open source project and the thesis focused on mainly generating executable experiments part of it. On the other hand, the findings suggest that its capabilities will expand by enriching our formal models.

In our future research, we intend to concentrate on three main issues: First, we will add new features and their implementations to provide finer grained variants. These variants will create a more flexible environment for the true users. Secondly, additional translators will be developed to target the environments other than Kepler. Although Kepler's reuse abstraction (actors) is well suited to support systematic reuse, it is desirable to add another target environment that works in completely different paradigm (e.g. Matlab, Repast). Lastly, a provenance mechanism which is based on the proposed domain model is planned to be developed. It is vastly important to store the information about conducted experiments to deduce new pointers for users (e.g. an experiment environment which assists the user based on their or others' previous experiments). Thus, defining and adopting a formal way to support provenance will be of interest in the next stage of our research.

REFERENCES

[1] NIST, “NIST/SEMATECH e-Handbook of Statistical Methods.” http://www.itl.nist.gov/div898/handbook/, 2013. [Online; accessed 2014-03-11].

[2] FlightFox, “How do airlines set prices.” https://flightfox.com/tradecraft/how-do-airlines-set-prices. [Online; accessed 2018-04-27].

[3] B. Zeigler, *Theory of Modelling and Simulation.* A Wiley-Interscience Publication, John Wiley, 1976.

[4] C. Drummond, “Replicability is not reproducibility: nor is it good science,” in *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th International Conference on Machine Learning*, (Montreal, Canada), pp. 2005–2008, 2009.

[5] L. Yilmaz, “Reproducibility in M&S research: issues, strategies and implications for model development environments,” *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 24, pp. 457–474, Dec. 2012.

[6] S. Robinson, R. E. Nance, R. J. Paul, M. Pidd, and S. J. Taylor, “Simulation model reuse: definitions, benefits and obstacles,” *Simulation Modelling Practice and Theory*, vol. 12, pp. 479–494, Nov. 2004.

[7] D. G. Cople and E. S. Brick, “A simulation framework for technical systems life cycle cost analysis,” *Simulation Modelling Practice and Theory*, vol. 18, pp. 9–34, Jan. 2010.

[8] M. D. Petty and E. W. Weisel, “Chapter 4 - model composition and reuse,” in *Model Engineering for Simulation* (L. Zhang, B. P. Zeigler, and Y. laili, eds.), pp. 57 – 85, Academic Press, 2019.

[9] R. G. Sargent, “A perspective on fifty-five years of the evolution of scientific respect for simulation,” in *2017 Winter Simulation Conference (WSC)*, pp. 3–15, Dec 2017.

[10] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, “Kepler: an extensible system for design and execution of scientific workflows,” in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 2004.*, pp. 423–424, June 2004.

[11] D. Talia, “Workflow Systems for Science: Concepts and Tools,” *ISRN Software Engineering*, vol. 2013, pp. 1–15, 2013.

[12] I. Altintas, S. Purawat, D. Crawl, A. Singh, and K. Marcus, “Towards A methodology and framework for workflow-driven team science,” *CoRR*, vol. abs/1903.01403, 2019.

[13] S. Cohen-Boulakia and U. Leser, “Search, adapt, and reuse: the future of scientific workflows,” *ACM SIGMOD Record*, vol. 40, no. 2, pp. 6–16, 2011.

[14] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski, “Cool features and tough decisions: a comparison of variability modeling approaches,” in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pp. 173–182, ACM, 2012.

[15] K. Czarnecki, C. Hwan, P. Kim, and K. Kalleberg, “Feature models are views on ontologies,” in *Software Product Line Conference, 2006 10th International*, pp. 41–51, IEEE, 2006.

[16] D. T. Sturrock, “Tutorial: Tips for successful practice of simulation,” in *2015 Winter Simulation Conference (WSC)*, pp. 1756–1764, Dec 2015.

[17] M. R. Barker and N. B. Zupick, “A Clue, the Cash, the Commitment, and the Courage: The Keys to a Successful Simulation Project,” in *Proceedings of the 2016 Winter Simulation Conference*, WSC ’16, (Piscataway, NJ, USA), pp. 80–87, IEEE Press, 2016.

[18] R. G. Sargent, “An Introductory Tutorial on Verification and Validation of Simulation Models,” in *Proceedings of the 2015 Winter Simulation Conference*, WSC ’15, (Piscataway, NJ, USA), pp. 1729–1740, IEEE Press, 2015.

[19] A. M. Law, "A Tutorial on Design of Experiments for Simulation Modeling," in *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, (Piscataway, NJ, USA), pp. 66–80, IEEE Press, 2014.

[20] S. M. Sanchez and H. Wan, "Work smarter, not harder: A tutorial on designing and conducting simulation experiments," in *2015 Winter Simulation Conference (WSC)*, pp. 1795–1809, Dec 2015.

[21] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *IEEE Computer*, vol. 40, pp. 26–34, December 2007.

[22] W. D. Kelton, "Design of experiments: experimental design for simulation," in *Proceedings of the 32nd conference on Winter simulation*, pp. 32–38, Society for Computer Simulation International, 2000.

[23] T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, Mar. 2006.

[24] D. Waltemath, R. Adams, F. T. Bergmann, M. Hucka, F. Kolpakov, A. K. Miller, I. I. Moraru, D. Nickerson, S. Sahle, J. L. Snoep, and N. Le Novère, "Reproducible computational biology experiments with SED-ML–the Simulation Experiment Description Markup Language.," *BMC systems biology*, vol. 5, p. 198, Jan. 2011.

[25] G. a. Silver, J. a. Miller, M. Hybinette, G. Baramidze, and W. S. York, "DeMO: An Ontology for Discrete-event Modeling and Simulation.," *Simulation*, vol. 87, pp. 747–773, Sept. 2011.

[26] L. N. Soldatova and R. D. King, "An ontology of scientific experiments.," *Journal of the Royal Society, Interface / the Royal Society*, vol. 3, pp. 795–803, Dec. 2006.

[27] R. Ewald and A. M. Uhrmacher, "SESSL: A Domain-specific Language for Simulation Experiments," *ACM Trans. Model. Comput. Simul.*, vol. 24, pp. 11:1–11:25, Feb. 2014.

[28] J. Schützel, D. Peng, A. M. Uhrmacher, and L. F. Perrone, "Perspectives on languages for specifying simulation experiments," in *Proceedings of the 2014 Winter Simulation Conference*, pp. 2836–2847, IEEE Press, 2014.

[29] O. K. Maps, "Overview of research on model-driven engineering." `https://openknowledgemaps.org/map/f73dfdff06453bac7345e3b9840c5464`. [Online; accessed 2018-04-27].

[30] O. K. Maps, "Overview of research on design of experiment." `https://openknowledgemaps.org/map/13da82c00fd5ae812123cbddefbf9c28/`. [Online; accessed 2018-04-27].

[31] O. K. Maps, "Overview of research on domain-specific language." `https://openknowledgemaps.org/map/cd2aa4fc6728a7a9b33eb59baa9b417e/`. [Online; accessed 2018-04-27].

[32] O. K. Maps, "Overview of research on computer simulation." `https://openknowledgemaps.org/map/14f5c3bedfaa8293f147e906a9ae10ba/`. [Online; accessed 2018-04-27].

[33] D. Montgomery, *Design and Analysis of Experiments, 8th Edition*. John Wiley & Sons, Incorporated, 2012.

[34] J. Ledet, A. Teran-Somohano, Z. Butcher, L. Yilmaz, A. E. Smith, H. Oguztuzun, O. Dayibas, and B. K. Gorur, "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*, DEVS '14, (San Diego, CA, USA), pp. 27:1–27:8, Society for Computer Simulation International, 2014.

[35] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," technical report, Software Engineering Institute, Nov. 1990.

[36] W. D. Kelton and R. R. Barton, “Experimental Design for Simulation,” in *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*, WSC ’03, pp. 59–65, Winter Simulation Conference, 2003.

[37] T. Thüm, C. Kastner, S. Erdweg, and N. Siegmund, “Abstract features in feature modeling,” in *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC ’11, (Washington, DC, USA), pp. 191–200, IEEE Computer Society, 2011.

[38] I. A. Hubner, M. Oliveberg, and E. I. Shakhnovich, “Simulation, experiment, and evolution: Understanding nucleation in protein s6 folding,” *Proceedings of the National Academy of Sciences*, vol. 101, no. 22, pp. 8354–8359, 2004.

[39] D. Beuche, H. Papajewski, and W. Schröder-Preikschat, “Variability management with feature models,” *Science of Computer Programming*, vol. 53, pp. 333–352, 2004.

[40] K. Czarnecki and M. Antkiewicz, “Mapping features to models: A template approach based on superimposed variants,” in *Generative programming and component engineering*, pp. 422–437, Springer, 2005.

[41] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, “Featureide: An extensible framework for feature-oriented software development,” *Science of Computer Programming*, vol. 79, pp. 70–85, Jan. 2014.

[42] R. C. Gronback, *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.

[43] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. a. Lee, J. Tao, and Y. Zhao, “Scientific workflow management and the Kepler system,” *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 1039–1065, Aug. 2006.

[44] I. Lorscheid, B.-O. Heine, and M. Meyer, “Opening the ‘black box’ of simulations: increased transparency and effective communication through the systematic design of experiments,” *Computational and Mathematical Organization Theory*, vol. 18, pp. 22–62, Oct. 2011.

[45] K. E. Stecke and J. E. Aronson, “Review of operator&sol; machine interference models,” *International Journal of Production Research*, vol. 23, no. 1, pp. 129–151, 1985.

[46] A. Gibiansky, “Quadcopter simulation in matlab.” `https://github.com/gibiansky/experiments/tree/master/quadcopter`. [Online; accessed 2019-03-26].

APPENDIX A: XPERIMENTER XTEXT GRAMMAR

```
grammar io.github.odayibas.Xperimenter with org.eclipse.xtext.common.Terminals

generate xperimenter "http://www.github.io/odayibas/Xperimenter"

Model :
  (elements += Type)
;

Type:
  NormalType   FeaturedType
;

FeaturedType:
   #ifdef  feature = ID
  (featureContent += NormalType)
   #endif
;

NormalType:
  Experiment   Design   Simulation   StatAnalysis   NumberGenerator   Variable
;

Experiment:
   experiment  name = ID
   {
     ( desc  description = STRING  ; ) ?
      objective  objective = Objective  ;
      design  design = [Design]  ;
      simulation  simulation = [Simulation]  ;
        analysis  analysis = [StatAnalysis]  ;
        visual  visualization = VisualizationType  ;
        target  target=TargetType  ;
       ( workdir  workdir = STRING  ; ) ?
       ( timeout  timeout = INT  ; ) ?
    }
;

Design:
   design  name = ID  {
     method  method = SamplingMethod  ;
     varlist  (variables += [Variable] )  ;
    }
;

Simulation:
   simulation  name = ID  {
     modelFile  modelFilePath = STRING  ;
    ( modelType  modelType = SimModelType  ; )
    (ports += Port)
    }
;

StatAnalysis:
   statAnalysis  name=ID  {
    action = Callable
    }
;
```

```

Port:
  InPort   OutPort
;

InPort:
   inport  name = ID  :  variable = [Variable]  ;
;

OutPort:
   outport  name = ID  :  variable = [Variable]  ;
;

Callable:
  Method   Service
;

Method:
   method  type=StatAnalysisType  ;
;

Service:
   service  uri=STRING  ;
;

Variable:
   variable  name = ID  :  type = VariableType  group
     group = VariableGroup ( [  lowValue = INT  ,
     highValue = INT  ] )? ( gen  generator=[
     NumberGenerator])?  ;
;

```

```
NumberGenerator:
   generator  name=ID  {
     method  uri=STRING  ;
    ( seed  seed=INT  ; ) ?
   }
;

enum Objective :
  COMPARATIVE   SCREENING   RESPSURFACE
;

enum SimModelType :
  STATIC   CONTINUOUS   DISCRETEEVENT
;

enum VariableGroup :
  FACTOR   RESPONSE   NUISANCE
;

enum VariableType :
  BOOLEAN   INTEGER   FLOAT   STRING
;

enum SamplingMethod:
  RANDOMIZED   RANDOMIZEDBLOCK   FACTORIAL
     FRACFACTORIAL   CENTRALCOMP
;

enum StatAnalysisType:
  HYPOTESTING   ANOVA   MANOVA   CONFINTERVAL
;

```

```
enum VisualizationType:
  HISTOGRAM   SCATTERPLOT   BARCHART   DEFAULT
;

enum TargetType:
  KEPLER   R_SCRIPT
;
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Dayıbaş, Orçun

Nationality: Turkish (TC)

Date and Place of Birth: 25.02.1983, Denizli

Marital Status: Married

EDUCATION

Degree	Institution	Year of Grad.
M.S.	METU, Computer Engineering	2009
B.S.	Hacettepe University, Computer Engineering	2005

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2018 - ...	Havelsan Inc.	Engineering Mng.
2015 - 2018	Havelsan Inc.	Technical Lead
2014 - 2015	Onami Inc.	Founder, CTO
2010 - 2014	Aselsan Inc.	Senior Software Eng.
2009 - 2010	Université Nice Sophia Antipolis	Researcher
2005 - 2009	Aselsan Inc.	Software Engineer

PUBLICATIONS

International Journal Publications

- “On the use of model-driven engineering principles for the management of simulation experiments” *Orçun Dayıbaş, Halit Oğuztüzün, Levent Yılmaz,* 2018, Journal of Simulation, Pg. 1-13, Taylor & Francis Group

International Conference Publications

- “Models as self-aware cognitive agents and adaptive mediators for model-driven science” *Levent Yilmaz, Sritika Chakladar, Kyle Doud, Alice E Smith, Alejandro Teran-Somohano, Halit Oğuztüzün, Sema Çam, Orcun Dayıbaş, Bilge K Görür* Proceedings of the Simulation Conference (WSC), 2017 Winter, Pg. 1300-1311, IEEE Press

- “A hybrid transformation process for simulation modernization and reuse via model replicability and scenario reproducibility” *Joseph Ledet, Sema Cam, B Kaan Gorur, Orcun Dayibas, Halit Oguztuzun, Levent Yilmaz, Alice E Smith* Proceedings of the 2015 AlaSim Conference, Pg. 8, IEEE Press

- “Toward a model-driven engineering framework for reproducible simulation experiment lifecycle management” *Alejandro Teran-Somohano, Orçun Dayıbaş, Levent Yilmaz, Alice Smith* Proceedings of the 2014 Winter Simulation Conference, Pg. 2726, IEEE Press

- “Toward model-driven engineering principles and practices to support model replicability” *Joseph Ledet, Alejandro Teran-Somohano, Zachary Butcher, Levent Yilmaz, Alice E Smith, Halit Oğuztüzün, Orçun Dayıbaş, Bilge Kaan Görür* Proceedings of the 2014 Summer Simulation Multiconference, Pg. 6, Society for Computer Simulation International

- “Toward model-driven engineering principles and practices for model replicability and experiment reproducibility” *J.Ledet, A.Teran-Somohano, Z.Butcher, L.Yilmaz, A.E. Smith, H. Oğuztüzün, O. Dayıbaş, B.K. Görür* Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative 2014, Pg. 27, Society for Computer Simulation International

- “Kutulu: A Domain-Specific Language for Feature-Driven Product Derivation” *Orcun Dayibas, Halit Oguztuzun* Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual, Pg. 105, IEEE