MODEL MANAGEMENT FOR HYPOTHESIS-DRIVEN SIMULATION EXPERIMENT WORKFLOWS

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

BY

SEMA ÇAM

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

SEPTEMBER 2022

Approval of the thesis:

MODEL MANAGEMENT FOR HYPOTHESIS-DRIVEN SIMULATION EXPERIMENT WORKFLOWS

submitted by **SEMA ÇAM** 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, METU**	______________

Examining Committee Members:

Prof. Dr. Ahmet Coşar Computer Engineering, Çankaya University	______________
Prof. Dr. Halit Oğuztüzün Computer Engineering, METU	______________
Assoc. Prof. Dr. Ebru Aydın Göl Computer Engineering, METU	______________
Assoc. Prof. Dr. Aysu Betin Can Informations System, METU	______________
Assoc. Prof. Dr. Kayhan İmre Computer Engineering, Hacettepe University	______________

Date:05.09.2022

Name, Surname:	Sema Çam
Signature :	

ABSTRACT

MODEL MANAGEMENT FOR HYPOTHESIS-DRIVEN SIMULATION EXPERIMENT WORKFLOWS

Çam, Sema
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Halit Oğuztüzün

September 2022, 100 pages

With today's breakthroughs in computational science and engineering, research experts can now simulate a lot of experiments on computers. Experiment specification is aided by frameworks and support systems for reusability and reproducibility of scientific research, as well as domain-specific languages, domain models, ontologies, data models, statistical analysis methods, and other types of tools and assets with related formalisms. Despite this, most frameworks or support tools for experiment specification ignore hypotheses and lack a procedure based on properly stated hypotheses. The main issue with a lack of hypotheses in the experimental process is that an experiment's credibility and repeatability can be harmed by an erroneous or inadequate record. Furthermore, the diversity of models, metamodels, tools, and data for testing bring the need for Global Model Management (GMM). In that sense, GMM leverages documenting, sharing, reusability, and replicability of simulation experiments by employing Model-Driven Engineering methodologies.

This thesis demonstrates how to use GMM to facilitate simulation experimentation with explicit hypotheses as a scientific workflow and proposes an extension to the Simulation Experiment Description Markup Language (SED-ML) that involves explicit specification of the hypothesis targeted in the simulation experiment. A megamodel, or registry for models and metamodels, is created particularly to serve as a repository for managing the artifacts used in a simulation project. All steps of a simulation experiment, including specification, input data production, experiment execution, and output data analysis, are effectively addressed by the megamodel. Then, using case studies, the applicability of GMM to simulation experiments is demonstrated. GMM, in our view, provides a solid framework for managing both experiment assets and experiment processes.

Keywords: Design of Experiments, Global Model Management, Model Driven Engineering, Simulation Experiment Description Markup Language, Signal Temporal Logic

ÖZ

HİPOTEZE DAYALI SİMÜLASYON DENEYİ İŞ AKIŞLARI İÇİN MODEL YÖNETİMİ

Çam, Sema
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Halit Oğuztüzün

Eylül 2022 , 100 sayfa

Hesaplamalı bilim ve mühendislikteki günümüz atılımları sayesinde, araştırma uzmanları artık bilgisayarlarda birçok deneyi simüle edebilirler. Deney tanımı, bilimsel araştırmanın yeniden kullanılabilirliği ve yeniden üretilebilirliği için çerçeveler ve destek sistemlerinin yanı sıra alana özgü diller, alan modelleri, ontolojiler, veri modelleri, istatistiksel analiz yöntemleri ve ilgili formalizmlere sahip diğer araç ve varlık türleri tarafından desteklenir. Buna rağmen, deney belirtimi için çoğu çerçeve veya destek aracı, hipotezleri görmezden gelir ve uygun şekilde ifade edilen hipotezlere dayalı bir prosedürden yoksundur. Deneysel süreçte hipotez eksikliği ile ilgili temel sorun, bir deneyin inanılırlığının ve tekrarlanabilirliğinin hatalı veya yetersiz kayıttan ötürü zarar görmesidir. Ayrıca, test için model, metamodel, araç ve verilerin çeşitliliği, Küresel Model Yönetimi (KMY) ihtiyacını beraberinde getirir. Bu bağlamda KMY, model tabanlı mühendislik metodolojilerini kullanarak simülasyon deneylerinin belgelenmesi, paylaşılması, yeniden kullanılabilirliği ve tekrarlanabilirliğini arttırır.

Bu tez, bilimsel bir iş akışı olarak açık hipotezlerle simülasyon deneylerini kolaylaştırmak için KMY'nin nasıl kullanılacağını gösterir ve simülasyon deneyinde hedeflenen hipotezin açık bir şekilde belirtilmesini içeren Simülasyon Deneyi Açıklama İşaretleme Dili'ne (SED-ML) bir uzantı önerir. Megamodel, ve ya modeller ve meta-modeller için bir kayıt defteri, özellikle bir simülasyon projesinde kullanılan yapıtları yönetmek için bir depo olarak hizmet etmek üzere oluşturulur. Tanımlama, girdi veri üretimi, deney yürütme ve çıktı veri analizi dahil olmak üzere bir simülasyon deneyinin tüm adımları megamodel tarafından etkin bir şekilde ele alınır. Daha sonra vaka çalışmaları kullanılarak KMY'nin simülasyon deneylerine uygulanabilirliği gösterilmiştir. Bize göre KMY, hem deney varlıklarını hem de deney süreçlerini yönetmek için sağlam bir çerçeve sağlar.

Anahtar Kelimeler: Deney Tasarımı, Küresel Model Yönetimi, Model Tabanlı Mühendislik, Simulasyon Deneyi Tanımlama İşaretleme Dili, Sinyal Zamansal Mantık

To my sister

ACKNOWLEDGMENTS

Throughout my master's and Ph.D. years, I have received a great deal of support and assistance from my supervisor, Prof. Dr. Halit Oğuztüzün, whose teaching was invaluable in formulating the research questions and methodology. Additionally, I am particularly grateful for the guidance given by Drs. Levent Yilmaz, Alice Smith, Ebru Aydin Göl and Aysu Betin Can. I would also like to thank the research group both in Auburn University and METU including Drs. Joseph Ledet, Orçun Dayıbaş and Bilge Kaan Görür.

I would like to convey my heartfelt gratitude to my beloved sister, Gizem, to whom my dissertation is devoted, and to my parents, who have always patiently believed in me and prioritized their children's education. Finally, I would like to express my gratitude to Mustafa, my partner, for bearing with me throughout this long process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DSL	Domain-specific Language
GMM	Global Model Management
GMM4SE	Global Model Management for Simulation Experiments
MIASE	Minimum Information About a Simulation Experiment
OMG	Object Management Group
SED-ML	Simulation Experiment Description Markup Language
STL	Signal Temporal Logic
XML	Extensible Markup Language
XPR	Xperimenter for Simulation Experiments

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Theories and hypotheses regarding what makes a system operate or explain particular phenomena in terms of cause and effect relations are put forth in many scientific disciplines and engineering problems. With a set of goals and questions in mind, research experts undertake experiments to test hypotheses. The system's input variables are purposefully changed, and the consequent output values are assessed and measured. Experiments provide evidence as to whether or not offered ideas are correct [4]. In this respect, a scientific process entails the stages listed below:

1. Develop well-structured research questions about a problem and its origin.
2. Gather statistical hypotheses from the questions to answer them.
3. Generate some logical hypotheses whose consequences are in the form of expected behavior.
4. Design experiments to put the fundamental hypotheses about the phenomenon to the test.
5. Validate the experiment for relevance and dependability.
6. If needed, create computer simulations (computational methods for producing behavior from a model), run them, and analyze the results.
7. Examine the hypotheses along with the underlying assumptions for the correctness and, if required, modify the model, experiments, or predicted behavior.

With recent advances in computational science and engineering, particularly modeling and simulation technologies, experiments are increasingly being done on computers to avoid the dangers, if not the practical impossibility, of performing experiments in the actual world. Furthermore, simulation trials often involve less time, money, and effort. Nevertheless, the main practical issue we face is that gaining a better knowledge of real-world processes becomes time-consuming and error-prone due to limitations of human perception. Because both simulation experiment designs and simulations themselves as software are frequently created by hand. As a result, assessing the quality of experiments becomes required, and the accuracy of the simulators that run the simulation models becomes a critical component of this assessment. In this sense, creating simulation models at a relevant degree of abstraction and accuracy, as well as planning trials with adequate information, would improve experimentation quality. A thorough record of the experimental settings enhances the credibility and repeatability of scientific experiments [5, 6]. Likewise, comprehensive experimentation practices necessitate establishing a clear and well-documented link between experiments, objectives, and hypotheses.

Nowadays, domain-specific languages (DSLs) and markup languages have had a significant impact on the academic debate on keeping accurate and adequate records of simulation trials. DSLs are frequently used to effectively generate experiment specifications and designs with a focus on a specific domain for development, as well as to enable recording and reusing of experiments for reproducibility reasons [7].

Markup languages are also emerging as a standard for simulation studies. Simulation Experiment Description Markup Language (SED-ML) [8] is an XML-based language for encoding, sharing, and documenting simulation experiments [9] to facilitate scientific collaboration. It makes it easier to share experiment descriptions, validate, and reuse simulation experiments. SED-ML standardizes the whole process, increasing the repeatability and trustworthiness of simulations. Despite the fact that SED-ML has many advantages for simulation experiments, hypothesis specification is not covered by the language, and we believe that adding the missing piece will significantly enrich the experimentation process.

When it comes to scalability, the concept of retaining a thorough record of simulation tests becomes even more intricate. Because there is an increasing number of simulation experiment projects, and these models require a supporting environment that is simple enough for non-programmers to utilize in order for the connected models to be sustainable and manageable.

The environment's experiment models, which are specifications for experiments specified by a DSL or markup language (e.g., a SED-ML specification), should be accessible and consistent with the other models, such as represented system or data. The environment must, in particular, facilitate loading, saving, editing, removing, archiving, searching, and executing models. Furthermore, models evolve in tandem with scientific pursuits. Their sophistication and diversity tend to grow with time. When many models are involved, users confront difficult system management concerns, such as the requirement to maintain consistency.

Model evolution necessitates new types of relation configurations (e.g., model to it's metamodel and from source to target model of a model transformation) among the other models. Furthermore, the system should be flexible, allowing it to be easily adapted to other areas. We acknowledge that simulation experimentation raises the following concerns:

- logical complexity at multiple levels, such as domain modeling (conceptual modeling) issues,
- model complexity,
- simulation complexity,
- heterogeneity of the operational environments for simulation execution,
- challenges in experiment design,
- difficulties in managing simulation input and output data, and other scenario-related data,
- challenges about the presentation of results (including visualization).

1.2 Contributions and Novelties

The concept of Global Model Management (GMM) [10] is offered to tackle the model management concerns in our study by facilitating megamodeling [2], i.e., fundamentally managing models and their relationships, for simulation experiments. GMM's objective is to manage an immense and diverse set of artifacts generated in modeling-in-the-large projects that use Model-driven Engineering (MDE) approach. Modeling-in-the-large refers to working with models, metamodels, and their attributes and relationships on a global scale. A megamodel also supervises and guides the simulation experimentation process in an MDE environment and guides scientists through simulation-driven tasks.

The goal of this study is to maintain a complete record of the experimentation process by introducing formally described hypotheses into the experimentation process by employing MDE techniques. The advantages of having formally described hypotheses are adding additional parametrization to the simulation experiments, enhancing the assessment of the experiments and testing of the hypotheses. Simultaneously, we strive to preserve consistency among the hypothesis and experiment models for sharing and recording the models across research scientists.

It is of interest to study if a generally recognized representation format for simulation experiments can be enhanced with a formal hypothesis definition. As a consequence, we offer a simulation experimentation approach based on an extended SED-ML with the hypothesis, which comprises accepting user-defined hypotheses with models as input, generating experiments to validate these hypotheses, and then executing and evaluating them. We essentially make explicit connections between hypotheses and experiments. We used an adaptive case study on the predictive analysis of hospital bed availability and congestion analysis on a traffic network as examples. These case studies also provide an opportunity to demonstrate the adaptability and effectiveness of SED-ML extended with a formally specified hypothesis in fields other than computational biology research. A successful model translation from SED-ML to a general purpose experiment model helps us to demonstrate the language's adaptability. In the later chapters, we will demonstrate the case study in details with a proposed hypothesis for the specified health institutions that is valid for predicting capacity fullness based on an increase in the number of COVID-19 patients. In the beginning, we create experiments by adding hypothesis definitions into an experiment model. Later, the Kepler Workflow Management System interprets the models and checks them with a trace checker. Finally, an analytical tool aids users in doing different statistical analyses.

1.3 The Outline of the Thesis

The remainder of the thesis is structured as follows. In Chapter 2, we present an overview of the existing works on the specification of simulation experiments, GMM, and megamodeling as well as a foundational background for the study. Chapter 3 Background starts with presenting Model-driven Engineering methodology with the Global Model Management concept, Signal Temporal Logic for hypothesis specification, Xperimenter and Regression Analysis, respectively. In the following Chapter 4, a hypothesis extension to SED-ML is introduced with the explanation of Hypotheses and Statistical Hypothesis Testing and Simulation Experiment Description Markup Language. Further, in Chapter 5, we explain our megamodel specification for Hypothesis-driven Experiment Design and explain Hypothesis-driven Experiment Design Workflow in Chapter 6 with user perspective in Chapter 7. Then, two case studies are performed for predictive analysis of hospital bed availability in Chapter 8 and congestion analysis on a traffic network in Chapter 9. Finally, Chapter 10 delivers conclusions and future work directions.

CHAPTER 2

STATE OF THE ART

Automating and recording experiments are significant for the reusability and reproducibility of scientific research [11]. Combining experimental design, model, and hypotheses in one process [12] with an appropriate relation improves experimentation practices. In this sense, there is a growing demand to standardize and automatize simulation experiments and provide accurate and sufficient records of simulation experiments [13]. This progress brought on the experimentation workflows as a response. Recently, Ruscheinski et al. [14] proposed an artifact-based workflow to examine the specified requirements and develop methods to accommodate goal-directed guidance to the user. Another experiment generation procedure is proposed to guide users through the specification process to hide away from the difficulties of the execution environment by Wilsdorf et al. [15], where they deploy schemas for abstraction. Similarly, Lorig et al. [16] and Chakladar et al. [17] have made seminal contributions to accomplish assistance systems embedding hypotheses into experiments and providing a process based on formally specified hypotheses. Lorig et al. [16] also presented a systematized process for simulation studies based on a formally specified hypothesis. Even though the reproducibility issue is well-covered by the current assistance systems for the entire life-cycle of an experimentation process, still, no experiment generation procedure or framework, to our knowledge, has considered integrating a broadly embraced standard computer-readable exchange format such as SED-ML to improve the usability of simulation experiments among experimenters and software tools in a globally managed megamodel environment. As well as promoting a standard with SED-ML, our method of megamodeling also contributes to the reusability concept by keeping the integrity of the simulation artifacts, i.e., any kinds of by-products produced during the development of simulation, in a fast-growing environment.

The use of formalisms for experiment design is salient to address the experiment specification and design, and reproducibility of the experiments [18]. In this sense, simulation experiment description languages such as Simulation Experiment Specification via a Scala Layer (SESSL) [19], have been designed. SESSL is an internal DSL embedded in Scala, helping to define and generating experiments. An experiment designed in SESSL consists of model specification, the description of replications, the stop state for the simulation run, the objective, and the range and optimization method. An alternative approach to the replicability intricacy is the ns3 Experiment Description Language (ENDL) [20] that captures experiment scenarios. It defines the design of an experiment in terms of factors, levels, and constraints that intend to exclude design points that are not required. However, it requires special-purpose tools for parsing and document validation. Perrone et al. [21] proposed SAFE Language for Experiment Description (SLED) to overcome NEDL's shortcomings. SLED utilizes a JavaScript Object Notation (JSON) format, which makes it much more straightforward to parse. Furthermore, dedicated markup languages, e.g., SED-ML [8] based on XML, pure functional programming languages [22], and ontologies have proven to be competent in specifying and managing simulation experiments. For this study, it is of interest to achieve the replicability of the experiments developed in diverse DSLs, ontologies [23], or other formal standards and yet continue being compatible with the scientific community.

The literature review has proven that numerous studies subsist as frameworks or assistance systems, and languages or formal standards to formulate and execute a simulation experiment. However, scarce authors have recognized the fact that current formalisms do not derive experiments incorporating hypotheses to enhance the reliability of the research [16]. This thesis initially fills this gap in research, so far lacking in the scientific literature. Specifically, we will provide a hypothesis-based experiment design process that takes system specifications, data sets, and the hypothesis concerning the system as inputs, and automatically generates an executable and validatable experiment as an output. With this experiment design process, we aim to solve the aforementioned simulation experimentation issues in the Introduction section by easing the burden of experimentation complexities at multiple levels. Additionally, the hypothesis specification is always carried along with the process so that no information is lost or wasted.

A problem rarely analysed in the previous workflow and process techniques are the reproducibility of the current simulation experiments among experimenters and a variety of software tools to provide backward compatibility. The question then becomes how completely to build a workflow that contains a variety of languages and tools, and therewithal receive an endorsement in the experimentation community. A possible solution to the issue at hand is integrating a standard experiment representation format such as SED-ML to our megamodel solution as a base artifact along with the other experimentation artifacts to increase the recognition. In this wise, the proposed process associate every user hypothesis to a generated experiment specification, then to an automatically produced SED-ML specification with the help of Model-Driven Development paradigm [24], and following the Model-Driven Architecture of the Object Management Group [25].

Then, a still-unsolved problem in the literature that becomes apparent is the scalability of the system proposed considering the dramatically growing interest in simulation experimentation. Previous studies have emphasized that GMM and megamodels are efficient to orchestrate a large number of software artifacts belongs to a particular domain [26]. Therefore, we see potential in GMM and megamodels concepts to undertake the scalability problem that may arise for the simulation experimentation process.

A megamodel represents the Model-driven Engineering artifacts, including transformation composition and an execution specification within a model. A diverse application field takes advantage from megamodeling for practical purposes such as a network functions virtualization (NFV) [27], a way for the agile deployment of network services for telecoms; the safety and security of software systems in critical domains [28]; an e-government project [29] and a software build system [30]. In particular, [27] defines a process for the design, deployment, and management of network services, and automating it to achieve benefits in cost reduction, reliability, and resilience for NFV systems. Nevertheless, to our knowledge, no prior studies have examined the simulation experiments domain with the guidance of a megamodel, and we propose to investigate whether a megamodel can steward numerous hypotheses and simulation experimentation artifacts in a workflow or not.

Briefly, unlike others, our study collects the standalone experimentation and hypothesis-based efforts into a megamodel and integrates these systems by model transformations to achieve a fully supported experimentation process based on SED-ML.

CHAPTER 3

BACKGROUND

Global Model Management and megamodel principles are utilized to create a customized hypothesis-based process design for simulation experiments while building and implementing our megamodel for the Hypothesis-driven experiment design process with a SED-ML extension. We employed and integrated many tools to realize the concept, including a formalism for hypothesis specification (STL), SED-ML, and a DSL for experiment design (Xperimenter). In this chapter, we will first describe the *Global Model Management* concept, as well as *megamodel* definition. Later in Chapters 5 and 6, we will demonstrate how they assist the implementation of our megamodel and how they aid in the integration of the phases of the experiment design process as a workflow. Then, we will explain *Xperimenter* and *Signal Temporal Logic*, and their specifications and usages will be shown in Chapters 8 and 9 with two distinct use cases. Afterward, in this chapter, we will introduce *Regression Analysis*, and how we employ a regression model will be given in Chapter 8 as part of our effort toward simulation data generation based on a regression model. Finally, *Kepler Scientific Workflow System* is presented as the last topic in this chapter, and its assistance for specifying our workflow will be given in Chapter 6.

It is worth mentioning that SED-ML and hypotheses and statistical hypotheses testing, as well as their purpose and significance in our work, will be addressed in Chapter 4.

For clarity and better understanding, in the following sections, we will distinguish and refer to models specifying an experiment as *experiment model* (e.g., SED-ML and Xperimenter models), specifications of system under study as *system model*, linear or polynomial regression model from which the simulation data sets are generated as *regression model*, and it will be referred to as *model* to define any type of Model-driven architecture model.

3.1 Global Model Management

Global Model Management (GMM) intends to assist *modeling in the large*. In a model engineering environment, the goal is to manage models, metamodels, and their attributes and relations. GMM is a sophisticated approach for creating, storing, viewing, accessing, modifying, and retaining the information connected with all of these modeling aspects. GMM [10] was introduced prior to the megamodel idea to give a model repository for changing numerous models. Other authors have since provided other modeling repositories, including REMODD [31] and MDEForge [32]. The primary goal of the research is to develop platforms for storing and acquiring artifacts with dependencies among the artifacts. However, due to the diversity of the artifacts, scalability remains one of the issues in model-driven engineering [26, 33, 34].

A model repository is a storage structure that stores model metadata and offers business logic that works on existing models. It allows for collaborative model modification and versioning. It acts as a bridge between the models and the user's activities such as searching, creating, editing/updating, and deleting. It allows you to query the models and keep track of the changes you've made to them.

GMM, as a Model-Driven Engineering environment, provides the following features:

- Large number of heterogeneous artifacts such as models, metamodels, model transformations, and source code.
- Modeling artifact relationships such as predefined (e.g., conformsTo) and ad hoc (e.g., weaving models).
- Tools for different purposes such as Model to Model (M2M) transformation engines, compilers, and workflow engines for simulation experiments.

Basic elements of GMM and their relations are shown in the Figure 3.1 [35]. In the figure, a megamodel is defined as collection of elements. An *Element* can be

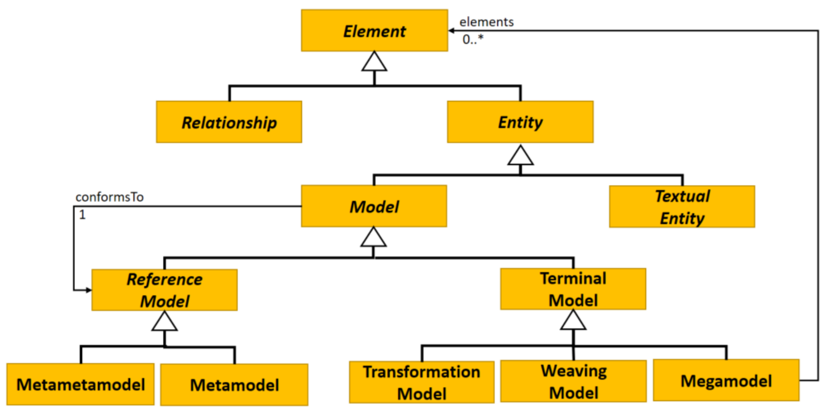


Figure 3.1: GMM conceptual framework from

an *Entity* or a *Relation*. An *Entity* can be a *Model* or a *Textual Entity*. MDE approaches delineate three kinds of models: *Terminal Models (M1), Metamodels (M2)* and *Metametamodels (M3)*. *Terminal models (M1)* conform to metamodels and they are the representations of actual systems. Furthermore, a *Terminal Model* captures the characteristics of a system and gives information about it. There exists a conformance relationship between a terminal model and its metamodel. *Metamodels (M2)* conform to metametamodels, and define domain-specific concepts. As for *Metametamodels (M3)*, they are at the top of the conformance hierarchy and provide generic concepts for metamodel specification. Finally, *Textual Entity* is a free-format entity that does not necessarily conform to any metamodel definition.

There are three kinds of Terminal Models: *weaving models, transformation models* and *megamodels*. A megamodel, being a terminal model, conforms to a specific metamodel: the metamodel of megamodels. Additionally, the model transformation relationship allows specifying the source and target reference models of a given transformation model.

3.2 Megamodel

A megamodel is still another model that incorporates models and many sorts of relationships between them. A megamodel is a collection of Model-driven Engineering artifacts such as model, metamodel, transformation, and any combination of these terms, such as transformation model, model transformation, and metamodel for transformation itself. According to [36], a megamodel is a model that contains other models as components. Stevens et al. further stated their point of view as the collection of models relevant to a system, and the interactions between them, may itself be considered as a model, which may demand and repay specific attention as a planned artifact [30]. They referred to this created item as a megamodel. In a nutshell, a megamodel is a metadata repository that holds exact representations of models and their relationships.

Users of the Megamodel repository can do a variety of operations, including searching for a model, registering newly produced artifacts, batch processing, and merging model pieces to extract data. When complicated criteria are involved, finding a model is a more difficult process than other tasks. After retrieving the models by searching, users may choose to analyze them by extracting and merging certain of their parts. The number of metamodels used by a transformation, or the number of models engaged in model weaving, are examples of metrics that may be acquired from the model repository.

3.3 Xperimenter for Simulation Experiments

Xperimenter is a DSL for simulation experiment design [37]. The DSL has three main objectives:

1. to serve as a platform for describing simulation experiments.
2. by linking parts of an experiment specification to higher-level abstractions, to manage simulation experiment variability (features).
3. to perform a simulation on a target platform, such as a scientific workflow management system.

Figure 3.2 shows a simplified metamodel definition of Xperimenter to provide a conceptual understanding of the components of a simulation experiment as well as the relationships between them.

- **Experiment:** The attributes of this class include information related to an experiment, such as the experiment name, date, and description. It may also include other information about the experiment, such as the cost of the experiment and the name of the investigator. The fundamental components of an experiment are the simulation model, the objective, the experiment runs, the design, the design matrix, statistical analysis, and visualization tools.
- **SimulationModel:** It's a crucial component of the simulation experiment. Identifying the experimental unit that delivers sample data is the first phase of the entire procedure in traditional experiments. In simulation experiments, on the other hand, the simulation model is the major source of data.
- **Objective:** The experiment's goal is defined by the class. The sort of experiment and the number of runs necessary to meet the experiment's aims are influenced by the objective.
- **Run:** Each run has a beginning and an end time. The number of experiment runs is determined by the progress of the experiment.
- **Design and DesignMatrix:** The experiment's structural component is maintained by the design class. The experimental structure is defined by the answers, factors, factor levels, and values, which are the mappings of the variables given by the user. The design determines a design matrix, which specifies the actual experimental runs, or the combination of factor values being investigated. Each row of the matrix corresponds to a factor level combination, and the execution fills the cells for the replies while the experiment is running.
- **StatAnalysis:** The response values and even the list of important factors are valuable in and of themselves, but without additional analysis, they are only of limited utility to an experimental. As a result, a statistical examination of this data can yield a plethora of helpful information and knowledge about the impact of various factors on response patterns. Hypothesis testing and confidence intervals are included in Xperimenter.
- **Visualization:** This fragment of the model denotes the method of visual representation of the analyses. Variable (Response and Factor): Variables are distinguished by their names and types. The metamodel currently supports four types

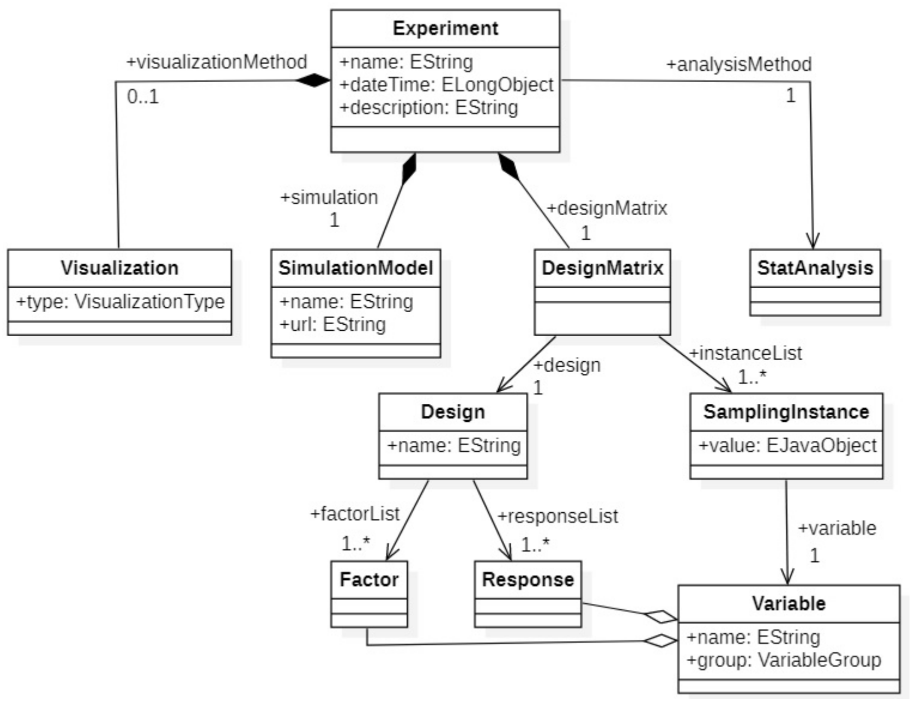


Figure 3.2: Simulation Experiment Domain Model

of variables (Integer, Float, Boolean, and String). Responses and factors are two separate types of variables, with responses representing the experiment's output values and factors representing the independent variables. Diversifying these factor values and documenting the results is an experiment. Each factor can have numerous levels (treatments), each having a one-to-one mapping between the factor level and the factor value. Each experimental run creates response values with a set of input parameters, and the simulation model consumes factor values. The factor values for that run are formed by the input parameters, and the inclusion of these values creates a factor-level combination. Each of the factor-level combination values matches a row of the design matrix, and the design matrix records the response values.

- **SamplingInstance:** An aggregate of a variable and its actual value is a sample instance. It is either a model input (factor variable) or a model output (output variable) (response variable). The sample instances can also be used in a design matrix.

3.4 Signal Temporal Logic

Signal temporal logic (STL) is a formalism that extends the linear temporal logic [38], and it is a system of rules and symbolism for representing real-valued signals [39]. An STL formula is composed of the Boolean and temporal operators and predicates in the form of linear inequalities. There are several practical usages such as runtime verification [40], and analysis of time series data [41]. In this study, we focus on the past time fragment of STL called ptSTL, where only the past time temporal operators are allowed, to specify our hypotheses with a formal method as user input. Hence, the user is fully responsible from the syntax of the hypothesis specification.

3.4.1 Past Time Signal Temporal Logic

In our opinion, ptSTL has the potential to represent the dynamic simulation models [42], where the state variable changes with respect to time (e.g., a car moving through a road). Thus, we made use of ptSTL formulas to correspond to the time course simulation experiments, defined in SED-ML for the continuous time systems. A ptSTL formula is defined with the following grammar in Equation 3.1 [3]:

$$\phi = \mathbf{T} | x_i \sim c | \neg\phi | \phi_1 \wedge \phi_2 | \phi_1 \ \vee \phi_2 | \phi_1 \mathbf{S}_{[a,b]} \phi_2 | \mathbf{P}_{[a,b]} \phi | \mathbf{A}_{[a,b]} \phi \tag{3.1}$$

According to Ergürtuna and Göl, in the grammar, a signal variable is represented by x_i, where $\sim \in \{<, >\}$ and *c* is a constant. The letter T represents the Boolean constant *true*. The standard Boolean operators are represented by $\neg$, $\wedge$ and $\vee$. Also, the temporal operators with time interval *[a, b]* are represented with $S_{[a,b]}$ *(since)*, $P_{[a,b]}$ *(previously)*, and $A_{[a,b]}$ *(always)*. Finally, the semantics of a ptSTL formula is defined over a signal for a given time interval.

For example, consider the ptSTL formula in Equation 3.2:

$$P_{[0,2]}(x_0) > 10, S_{[0,10]}(x_1) < 30 \tag{3.2}$$

The specification states that within the last 10 seconds, x_1 goes under 30, and since then, within every two seconds, x_0 goes above 10.

3.5 Regression Analysis

Regression analysis is a predictive modelling technique which investigates the relationship between dependent and independent variables. This technique is commonly used for finding cause- effect relationship between the variables. Although the earliest form of regression was published by Legendre in 1805 and by Gauss in 1809, regression methods still continue to be an area of active research. Therefore, many techniques for carrying out regression analysis have been developed. However, we will only examine *Linear* and *Polynomial Regression*s. Before explaining the Regression methods, it is important to explain surrogate modeling and how it is related to Regression Analysis. Surrogate models are constructed by using a data-driven approach where only input-output behavior of the simulation is assumed to be known. Limited number of carefully chosen data points are used for modeling the response of the simulation. In this context, Regression Analysis is one of the example of surrogate modeling where available data is utilized to model the relationship among the variables. We utilize Linear and Polynomial Regression, later to be explained, as our model to generate data for our system under study.

3.5.1 Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. A linear regression line has an equation of the form

$$Y = a + bX$$

where X is the explanatory variable and Y is the dependent variable.

3.5.2 Polynomial Regression

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial. The equation of the form

$$Y = a_0 + a_1X + a_2X_2 + ... + a_nX_n$$

represents polynomial regression.

3.6 Kepler Scientific Workflow System

A scientific workflow involves a series of structured computations or data manipulation steps that arise in scientific problem-solving. Similarly, simulation experiments are designed step by step in a computer environment to show that proposed theories are supported or not. The workflow systems mostly support graphical user interfaces to combine different technologies with efficient methods to increase the efficiency of the scientists. There are many examples of scientific workflow management systems. Kepler [43] is one of the software platforms for designing and implementing scientific workflows which we opted to use to design and realize our workflow. The tool provides processing and monitoring data, provenance information, and high-speed data movement. The system defines workflows as directed graphs where the nodes represent discrete computational components, and data and results can flow through the edges. Kepler defines nodes as *Actors* and edges as *Channels*. The scientific workflows model the flow of data from one *Actor* to another to achieve a goal. The tool provides a graphical user interface, a runtime engine, and a distributed computing option.

CHAPTER 4

A HYPOTHESIS EXTENSION TO SED-ML

One of the visions of this thesis is to address the issue of experiments having a lack of hypothesis specification and how the addition of such information can increase the credibility of scientific research. Having this purpose in mind, we believe that adding a hypothesis specification to a widely used and acknowledged simulation specification language, i.e., SED-ML, will significantly improve the experimentation process. Therefore, we explore the idea in this chapter by explaining SED-ML, hypotheses, and statistical hypothesis testing concepts first then, the chapter continues on how these concepts are incorporated into SED-ML as a hypothesis extension.

4.1 Simulation Experiment Description Markup Language

The Simulation Experiment Description Markup Language (SED-ML) [8] is an XML-based standard for expressing simulation descriptions on computer models of biological systems. Its goal is to save data from a simulation experiment performed on one or more models with a specific set of inputs. As the number of computational models, their size, and complexity continue to grow at an alarming rate, the necessity to improve on prior research by reusing models becomes increasingly apparent. Surprisingly, multiple initiatives to standardize the representation of computer models in different areas of biology raise the demand for model interchange and reuse. SED-ML intends to solve this need by providing reusable and inter-changeable models.

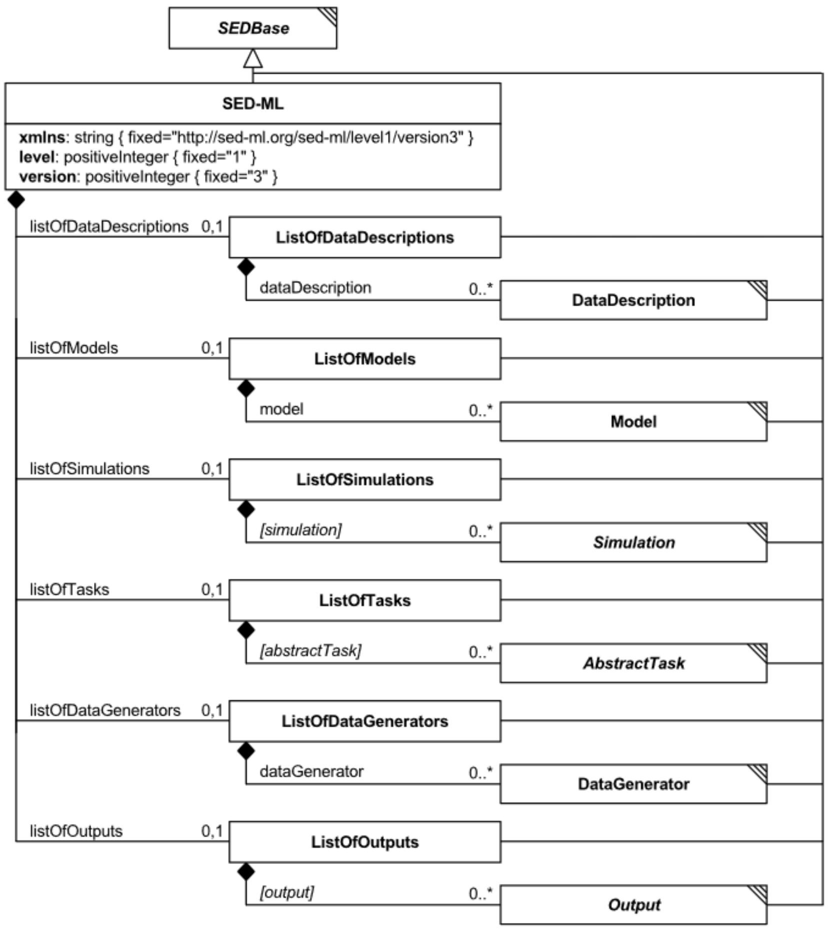


Figure 4.1: The SED-ML Class UML Class Diagram [1]

The SED-ML format in Figure 4.1 consists of six primary blocks:

1. **DataDescription** entity specifies the datasets for a simulation experiment.
2. **Model** entity is a reference for the models used in a simulation experiment, and the entity defines the procedures that are executed before the simulation (e.g., changing the value of an observable). Each instance of the Model class has a unique and mandatory id.
3. **Simulation** entity assists in the execution of the defined algorithm(s).
4. **Task** entity serves for a single simulation at a time. An experiment description can have as many tasks as required. The tasks do not require a specification for ordering.
5. **DataGenerator** entity produces post-processing procedures and applies those procedures to the simulation result before achieving the output. The post-processing steps can include mathematical manipulations such as normalization of data or mean-value calculation.
6. **Output** entity specifies the simulation output.

A typical SED-ML workflow is shown in Figure 4.2. Firstly, a SED-ML simulation experiment is initialized by creating a new SED-ML file. Then, models of the simulation are specified and saved into the created SED-ML file. Afterwards, simulation experiment setups are specified and saved into the same file. For assigning a setup to a number of models in the experiments, tasks are defined and recorded. After simulation is executed, outputs are obtained according to specified tasks and performed simulation experiment. The output is also added to the SED-ML file and whole simulation experiment is preserved in the file.

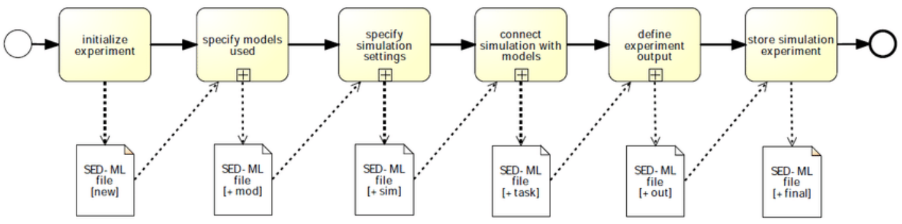


Figure 4.2: The process of defining a simulation experiment in SED-ML [1]

4.2 Hypotheses and Statistical Hypothesis Testing

A hypothesis is a formal assertion about the status of a natural population that presupposes relationships between variables related to a topic of study [44, 45]. In statistics, hypotheses are tested supposing that the probabilities are distributed over the values of the variables from the observed data models, [46], and the hypothesis testing involves two statistical hypotheses: **null (H0)** and **alternative (H1)** hypotheses, [47]. In general, a null hypothesis is a hypothesis to be tested, whereas an alternative hypothesis is an alternative to the null hypothesis that is supported if the null hypothesis is not plausible. Following the definition of the hypotheses, a formal hypothesis testing technique is used to evaluate whether to reject a null hypothesis. Hypothesis testing can result in two sorts of errors: **Type I Error** and **Type II Error**.

- **Type I Error** occurs if a true null hypothesis is rejected. The probability of making this error is defined with a *significance level* [48].

- **Type II Error** occurs if the researchers fail to reject a false null hypothesis.

There are several sorts of statistical tests, and many aspects influence their selection, such as the quantity and level of data, or the statistics used in the study [49]. However, in the limitations of this study, we merely extended the SED-ML with null and alternative hypothesis specifications. We recognize that additional effort is required to expand our hypothesis definition with more statistical testing characteristics, such as significance level.

4.3 Metamodel for the Extended SED-ML

For the purpose of extending the SED-ML with a hypothesis, a new block for specifying hypotheses called **Hypothesis**, is appended to the SED-ML specification. With the added hypothesis specification, the number of SED-ML primary blocks given in Section 4.1 increased to seven. Beside representing features of a hypothesis, our hypothesis specification also includes additional properties to conduct a statistical hypothesis test.

SED-ML with hypotheses extension serves as the intermediate experiment model in this work for our model transformations, notably the SED-ML to Xperimenter model transformation. The benefit of having an intermediate experiment model is that the approach improves the dependability of the generated models by allowing replication and validation of the intermediate model itself. In addition, the intermediate model is essential for capturing the characteristics of the experiments such as model, design, and, now hypothesis of an experiment.

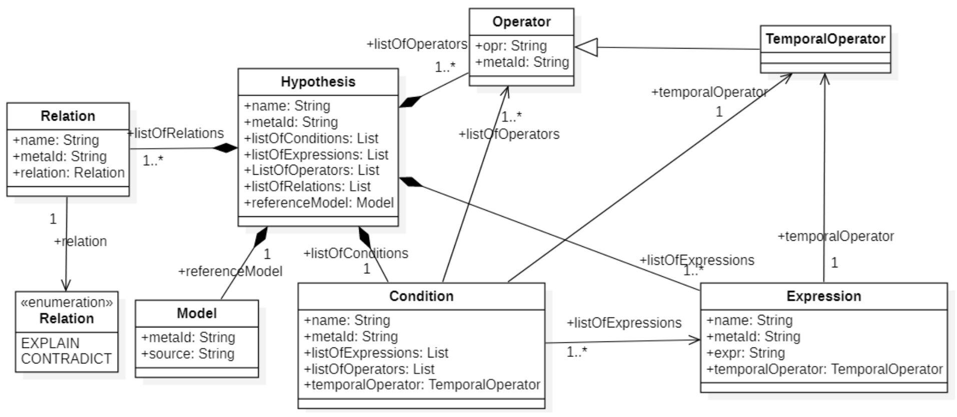


Figure 4.3: Hypothesis Extension for SED-ML Metamodel

Figure 4.3 displays a simplified metamodel description of the statistical hypothesis extension for the SED-ML. The complete metamodel specification including all the primary blocks of SED-ML can be found in Appendix A. The entity **Hypothesis** which is the main class in the diagram consists of either a single hypothesis or composite hypotheses with a relationship between them. The composite hypotheses enables the statistical hypothesis testing by including the both *null* and *alternative* hypotheses.

The **Hypothesis** block in SED-ML consists of one or more conditions with numerious expressions representing the STL equations and the expression relations are indicated with logical operators, e.g., *and*, *or*. Notably, each *listOfExpressions* block can include several *expression*s with multiple *operator*s and a *temporalOperator*. In addition, each *condition* can have a single *temporalOperator*. A *temporalOperator* states *T, P, S,* and *A* abbreviations within a time interval, e.g., P[1, 1]. The details of the temporal operators can be found in the Section 3.4.1. Also, the *listOfRelations* indicates the relations among the hypotheses. The relations between the hypotheses are shown with a *relation*. For example, if there is a single null hypothesis with multiple conditions that lead to the hypothesis expression, then, *EXPLAIN* relation is used . Or, if a null hypothesis is going to be tested with an alternate hypothesis, then *CONTRADICT* realtion is used. Finally, a *modelReference* associates the *expression*s of the hypothesis to the *model* block of SED-ML.

Our metamodel specification for SED-ML extended with hypothesis specification is open to extension with more features from the hypothesis specification and additional statistical hypothesis testing techniques. However, we concluded that the given specification is sufficient to perform our case studies. To conclude, Chapters 8 and 9 for case studies include the realization of SED-ML models, including the hypothesis entity **Hypothesis**.

CHAPTER 5

A MEGAMODEL FOR HYPOTHESIS-DRIVEN EXPERIMENT DESIGN

Experiments are being done on computers to avoid the danger, or perhaps the practical impossibility, of performing experiments in the actual world, thanks to recent advances in computational science and engineering, modeling, and simulation technologies. *In silico* experiments are a type of such experiment. To explain briefly, simulation experiments often take less time, money, and effort. Therefore, experimental scientists and engineers frequently employ simulation experiments and there are an increasing number of simulation experiment initiatives, such as myExperiment [50] and Experiment [51], which are basically social websites enabling academics to share scientific procedures. The development of numerous types of simulation experiment models is a part of these programs. However, in order to be sustainable and maintainable, these models require a supportive environment that is simple to use by non-programmers. As models evolve in tandem with scientific research, over time, their intelligence and diversity tend to grow. Users are confronted with serious system management challenges when a large number of models are involved, including the necessity to maintain consistency. During model evolution, new types of model relationships can be configured. The system should also be easily adaptable to new areas and abstract away the simulation experimentation complexities introduced in Chapter 1. The environment's experiment models should be available and manipulable (e.g., loading, saving, editing, deleting, searching, and running models). Global Model Management (GMM) is an appropriate idea which we study for this challenge in this regard. GMM seeks to handle a wide and diverse set of artifacts created in *modeling-in-the-large* (globally dealing with models, metamodels, and their attributes and relations [52] efforts in Model Driven Engineering projects (MDE).

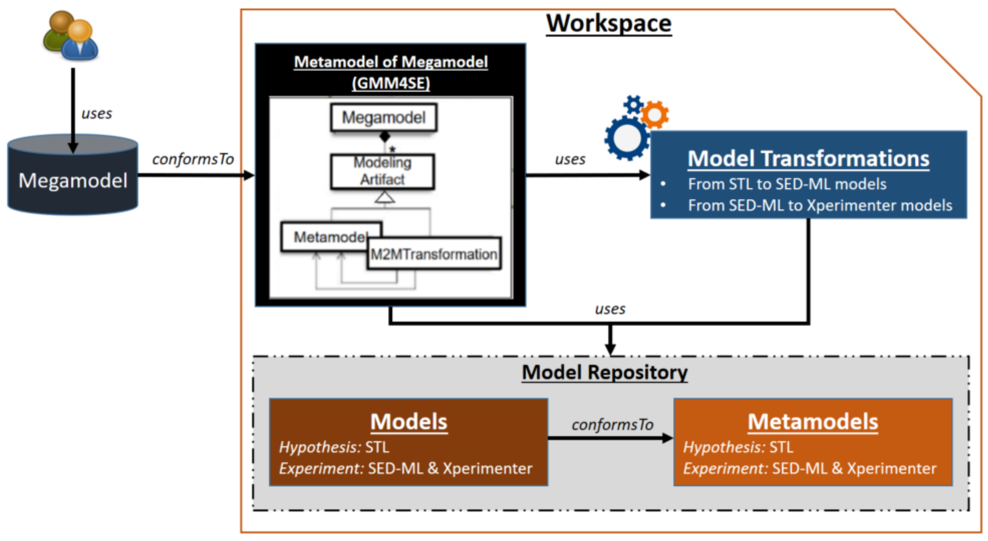


Figure 5.1: Overview of Megamodel for Simulation Experiments

One of the goals of this thesis is to give a design approach for experimental scientists to follow when constructing an MDE-based ecosystem. The GMM idea guides us to the administration of such a complicated system in this direction. As a result, we created a megamodel that encapsulates the modeling artifacts of scientific experimentation methodologies, with the necessary groundwork necessitating the creation of a metamodel for simulation experiment megamodels. The advantage of adopting a megamodel is that conforming megamodels can be developed and expanded in time by experimental scientists for their specific studies. The megamodel is known as GMM4SE [2], which stands for Global Model Management for Simulation Experiments. Metamodels, transformations, and relations between metamodels and model transformations are all supported forms of modeling artifacts in GMM4SE. The artifacts represent data, transformation, or model relationship (e.g., the dependency between the source and target models of model transformation) for a certain megamodel, and the user interacts with them as a coherent unit. The megamodel adhering to the GMM4SE metamodel is depicted simply in Figure 5.1. The workspace in Figure 5.1 refers to the model workspace, which structurally contains the modeling artifacts (e.g., a metamodel on top of multiple models).

Since our goal is to achieve a Hypothesis-driven Experiment Design process which will be explained in Chapter 6 in detail, we incorporated frameworks, DSLs, and a workflow engine that are serving for the experimentation domain into the megamodel.

Our megamodel, as proof of concept, includes a model-based approach that includes three types of modeling artifacts:

1. A metamodel for the megamodel that captures the specifics and dependencies of the modeling product.
2. Metamodels and models of the languages for experiment specification (e.g., STL, Xperimenter for Simulation Experiments, SED-ML),
3. Metamodels and models of the workflow management engines for experiment design, capturing the steps for creating, executing, and analyzing the experiments (e.g., Kepler Workflow Management System).

5.0.1 Construction of GMM4SE Megamodel for Simulation Experiments

The GMM4SE megamodel requires three main components in order to provide the aforementioned utilities:

1. A workspace for storing the modeling artifacts,
2. A metamodel specification for GMM4SE,
3. Model interfaces for user operations such as model loading and model querying.

Figure 5.2 depicts the components of the GMM4SE megamodel and the available user operations. The first component of the GMM4SE megamodel is merely a directory on a hard drive. The metamodels, models, and model transformations artifacts are all separately located on the hard drive. The second component of the GMM4SE megamodel is already explained in the previous section of this chapter. Therefore, we find it more useful to focus on the last component: *model interfaces* of GMM4SE megamodel. The *model interfaces* component seeks to provide a separate layer to load, edit and delete models, query the available models, create links among them and apply model transformation from/to available models. Also, the *model interfaces* layer aims to separate the megamodel from user interfaces. This separation leads to a flexible and reusable model management environment.

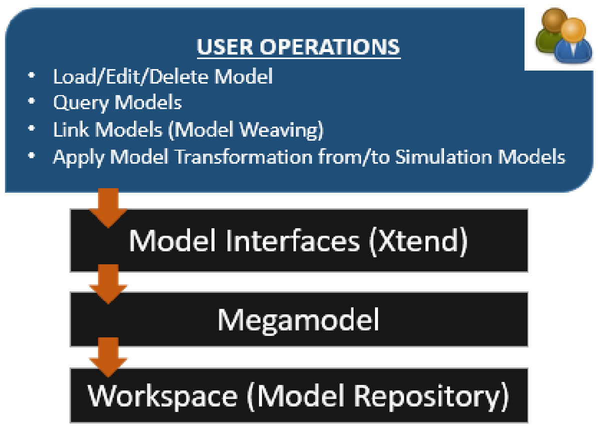
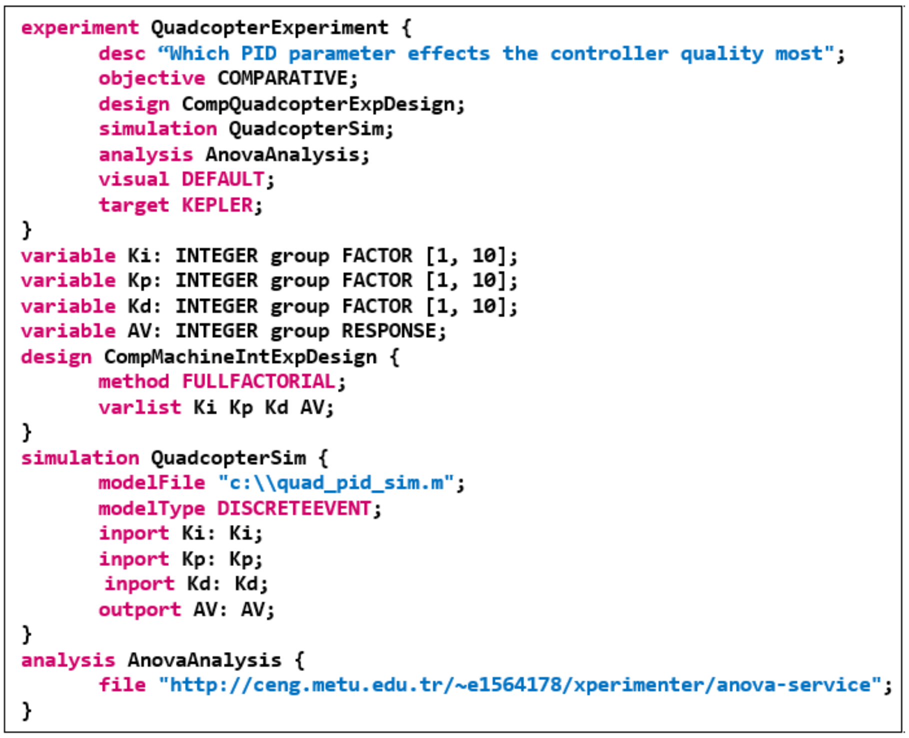

USER OPERATIONS
Load/Edit/Delete Model
Query Models
Link Models (Model Weaving)
Apply Model Transformation from/to Simulation Models
Model Interfaces (Xtend)
Megamodel
Workspace (Model Repository)

Figure 5.2: Megamodel Structure including user operations

5.0.2 Case Study: Quadcopter

A quadcopter experiment [2] is modelled with Xperimenter to clearly demonstrate how the megamodel interfaces functions. We consider that a multi-rotor helicopter managed by a flight control system serves well for this purpose. In that sense, a multi-rotor helicopter is a quadcopter with four rotors that lifts and propels the helicopter. The quadcopter usually has a quadcopter flight control system, and the system is a worthy illustration of how a Proportional-Integral-Derivative (PID) controller modifies operational variables to maintain an output variable at a given set-points. However, it might be risky and costly to verify the quadcopter controller through genuine test flights. Thus, we must create a practical method for validating a controller. PID control, as its name implies, uses three coefficients: proportional, integral, and derivative *(Kp, Ki, Kd0*). These coefficients are altered to get an ideal result.

The quality of a controller relies on the gain values and adjusting these parameters requires an expert's instinct and time. We think that Xperimenter DSL promotes this tuning well. Initially, we need to define a research question: *"Which gain parameter is most important to decide the quality of the controller?"* with an assumption that we have a differential equation model for quadcopter flight and configurable gain parameters. The Xperimenter model in Figure 5.3 describes a full factorial design for this particular experiment.



```
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 QuadcopterSim {
      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";
}
```

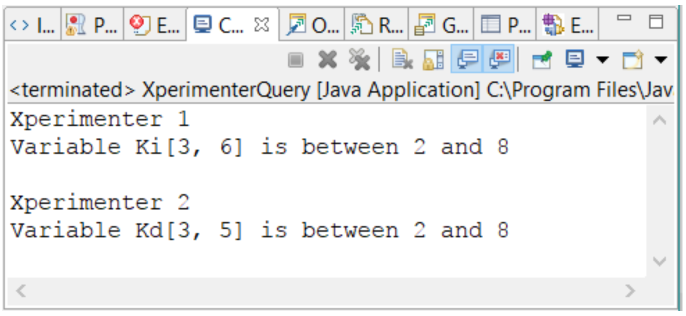
Figure 5.3: Quadcopter Experiment model [2]

As an example of what the Xperimenter model resembles in our workspace, the Quadcopter Xperimenter model in Figure 5.3 describing the design, simulation, and analysis of the experiment is given. There are four variables, called *Ki, Kp, Kd* and *AV*, in order. The experiment design is named as *CompMachineIntExpDesign* and the selected design style is *FullFactorial* that allows for estimation of the main effects and interactions of the aforementioned four variables. The model has a simulation element called *QuadcopterSim* that is operated on the variables *Ki, Kp, Kd* and *AV*. The model file and model type are also specified in the simulation element. Besides, the variables are distinguished as either inports or outports in the simulation. In this regard, an inport is a link to pull data from the outside of the simulation, and an outport is a link to push data to the outside. In the end, an Anova Analysis takes place and the Anova service is accessible from the given URL.

We designed a case study where only two Xperimenter models are used for the sake of simplicity. The models have similar specifications where the variables *Ki, Kp, Kd* and *AV* vary. The variables for the Xperimenter models are depicted in Table 5.1.

Table 5.1: Quadcopter Models Input Variables [2]

	name	lowValue	highValue
Quadcopter 1 Variables	Ki	3	6
	Kd	5	9
	Kp	6	8
Quadcopter 2 Variables	Ki	2	7
	Kd	3	5
	Kp	1	8



```
<terminated> XperimenterQuery [Java Application] C:\Program Files\Jav
Xperimenter 1
Variable Ki[3, 6] is between 2 and 8

Xperimenter 2
Variable Kd[3, 5] is between 2 and 8
```

Figure 5.4: Xperimenter model validation result [2]

The case study is based on a query where the lowest and highest values of the model input variables remain between a limit. To realize such a query, we made a deliberate chose for using Xtend language [53]. We found the Xtend language convenient as our metamodel specifications are created with Eclipse Modeling Framework [54]. In that sense, Xtend language provides an interface to register the Xperimenter metamodel, and the Xperimenter models are loaded as resource sets. From these resource sets, the content of the models becomes reachable as Xperimenter components (e.g., experiment, simulation, and design). In our particular query example, we perform a query where the input variables of the models have the lowest value that is bigger than *2* and the highest value that is smaller than *8* (i.e., *lowValue* > *2* and *highValue* < *8*). Finally, the query result is achieved in Figure 5.4 and the query implementation by using Xtend language can be found in B.

CHAPTER 6

A HYPOTHESIS-DRIVEN EXPERIMENT DESIGN WORKFLOW

The proposed workflow for the hypothesis-based simulation experiments consists of three phases: **pre-execution** (i.e., input preparation), **execution** and **post-execution**. The main components considered as part of the workflow are the execution and the post-execution steps, and they are only triggered after taking the appropriate user inputs. During the pre-execution phase, the user prepares the input to translate the system under study, data sets that comply to a regression model, and a hypothesis to understand the system behavior. User inputs are limited to the megamodel registered members (e.g., STL, SED-ML and Xperimenter); however not limited in the sense of extendability. For example, even though only STL exists as the registered hypothesis specification formalism, the options can be extended with other formalisms and their transformations as an artifact, later. Likewise, we have not a single, but two different experiment specification languages exist in our megamodel (i.e., SED-ML and Xperimenter) and can be extended with more languages. Hence a user is also capable of extending the megamodel with desired hypothesis formalisms and experiment models as long as providing their metamodel and model transformations.

After completing the pre-execution phase, hypothesis-driven experiment design workflow takes place and initiates the execution phase including the model transformations. The GMM4SE megamodel with artifacts is employed as the main actor of the workflow in order to find the relations and the model transformations between the hypothesis and experiment models. In this sense, the first goal of the execution phase is generating an executable experiment model from the hypothesis and then, transforming the experiment model into different formalisms. The second and main goal of the phase is to execute the experiment and achieve an output. Note that the orchestration

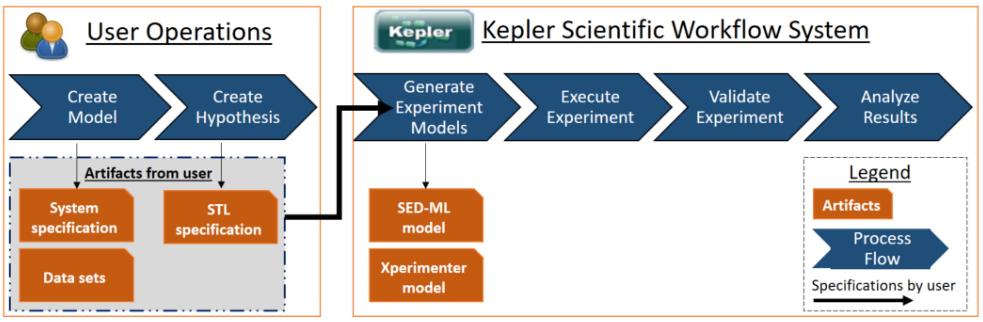


Figure 6.1: The workflow for Hypothesis-driven Experiment Design process

of the tasks is fulfilled by a scientific workflow management system (i.e., Kepler).

The final phase of the workflow (i.e., post-execution) begins after executing the experiment and achieving a throughput. The throughput of the experiment indicates if the hypothesis is valid or not. The tasks are not limited to revealing the result, though. A trace analysis technique is used to verify the hypothesis specification and to find if any violations exist. Then, an optional statistical analysis tool is provided to achieve a full flow of the scientific experimentation process. The post-execution phase is also governed by the scientific workflow management system (i.e., Kepler).

Figure 6.1 depicts the user operations and the workflow as two separate blocks yet connected and sequential blocks, respectively. In this sense, pre-execution is represented as the user operations whereas the execution and post-execution steps are represented as the workflow.

The steps of the user operations in the workflow are explained as follows:

1. For the system under study, the user constructs a system model and collects/-generates data sets for the described system. This system model includes input and output variables including their limits and intervals, as well as a tracing formula for marking the model's time traces for the specified hypothesis. The input variables' values should be included in the data sets. Chapter 8 contains more information on the system specification and the aptitudes of each module.

2. The user specifies a hypothesis for a system under study (e.g., hospital bed availability prediction), in a formally defined hypothesis language (e.g., STL

specification in Section 3.4.1).

Our process for Hypothesis-driven Experiment Design starts after the user completes the system and hypothesis specification. The workflow coordinates the model transformation to generate an experiment from the supplied hypothesis, as well as the execution, validation, and analysis of the generated experiment step by step.

Our workflow for Hypothesis-driven Experiment Design, defined as a Kepler workflow, is given in Figure 6.2. The figure shows the same experiment execution process in Figure 6.1 as represented in Kepler scientific workflow system. The primary components in the figure that carry out the process are described as follows:

1. **Hypothesis 2 Experiment Transformator**: is in charge of converting a formal language into a specification for an experiment (specifically, from ptSTL to Xperimenter model transformation, and SED-ML). See Appendix C and D for the code specification.

2. **Experiment Executor**: converts the experiment specification into an executable, runs the executable experiment, and returns the throughputs (specifically, from Xperimenter to Kepler Workflow Management System). See Appendix E for the code specification.

3. **STL Trace Checker**: validates if the experiment's result corresponds to the user-defined hypothesis, and locates non-fitting time traces. Further details regarding the STL Trace Checker can be found in [3].

4. **Analysis Tool Executor**: provides statistical analysis techniques to the user, optionally. See Appendix F for the code specification.

6.0.3 Construction of Hypothesis-driven Simulation Experiment Workflow

Hypothesis-driven Simulation Experiment Workflow is a composition of consecutively connected several distinct modules via the Kepler Workflow Management System. The modules are independently implemented by using Python language. However, the modules are sequentially executed via Kepler executor actors in a workflow.

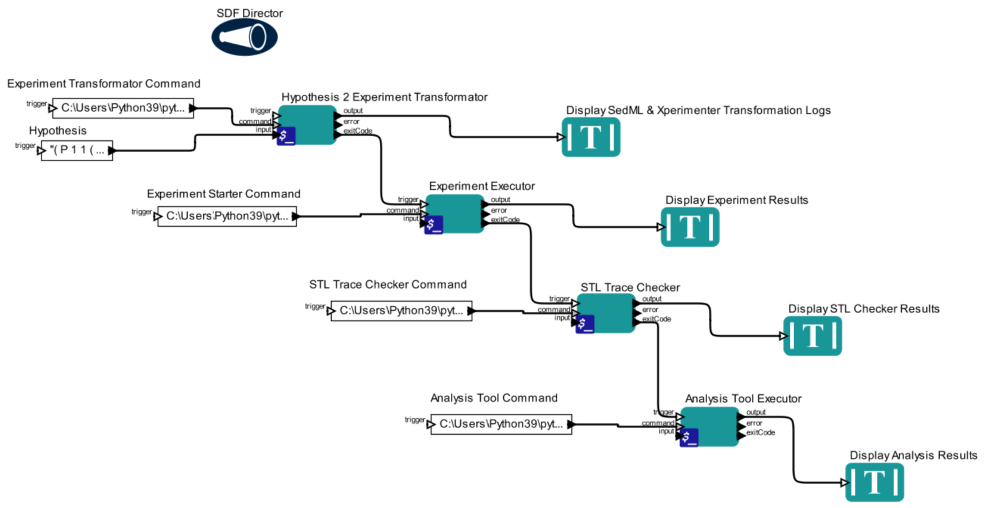


Figure 6.2: The Kepler workflow for hypothesis-based experiment design

As for the execution of the workflow, it is controlled by a director. In our workflow, the actors are instructed via Synchronous Dataflow (SDF) Director that executes a single actor at a time with one thread of execution [43]. The reason for employing the SDF Director rather than the other directors is that the SDF Director can become very efficient and does not cause overhead when system resources are in use. This efficiency is provided by precalculating the schedule for actor execution.

In our workflow, each executor actor acquires a constant command (e.g., *Experiment Transformator Command* and *Experiment Starter Command* in Figure 6.2) to locate and trigger the module with the Java and Python executables. After each sequential execution, the module displays its output visually and then triggers the next module to execute. The following shows an example of a command:

python.exe "ExperimentRunnerForHospital.py" "$\phi_4 = P_{[1,1]}(h1_{next} < 3048)$*"*

The command starts with a parameter that is the related executable of the preferred language, i.e., Python in our case. Then, the command continues with parameter that refers to a file directory where the code implementation resides. Finally, the last parameter of the command is an optionally passed argument to the code. Another way of passing an argument to the code is to use the input interface of the executor actor. An input can be passed to the workflow by the workflow user to increase user interaction. In our workflow, the hypothesis specification is received as an input argument to generate the SED-ML model.

CHAPTER 7

HYPOTHESIS-DRIVEN EXPERIMENT DESIGN WORKFLOW FROM USER PERSPECTIVE

This chapter aims to clarify the proposed hypothesis-based simulation experiments workflow from the user perspective. Each step that the user is responsible for and the meaning of the outputs from each module in the workflow will be scrutinized.

7.1 Pre-execution: User Operations

During the pre-execution phase, the user prepares three different inputs for the workflow:

1. Dataset generation from an authentic data
2. Hypothesis specification
3. System specification

7.1.0.1 Dataset generation from an authentic data

The user is expected to find authentic data and generate a dataset from it (e.g., daily COVID-19 patients in Turkey [56]). The dataset generation should be based on a regression model. The used regression model is referenced in the simulation models in our workflow (i.e., SED-ML and Xperimenter). Later, during the execution of the simulation, the datasets are validated for their compliance with the regression model.

In order to provide an example, following regression model from our case study regarding the hospital bed availability is given:

$$Y = b_1 X_1 + 0.5 X_0 + c$$

where X_1 is the *number of patients for the time step* that represents the share of COVID-19 of the city of Ankara, b_1 is the ratio of the hospital capacity to total capacity of the all hospitals in Ankara and X_0 is the capacity of the hospital. Finally, c represents the random number which is limited to the daily patients from neighbor cities.

The defined regression model is realized by Python language in the Listing 7.1.0.1 for a hospital (i.e., *h1*). In the code, first, the authentic data for daily COVID-19 patients in Turkey and hospital capacities are read into the data frames. Then, based on the regression formula, the dependent value y is calculated. Finally, the defined variables fit into the regression model.

```python
import numpy
import random
import pandas as pd

df_patients = pd.read_excel(r'cases_turkey.xlsx')
df_patients.columns = ["p"]

df_capacity = pd.read_excel(r'capacity.xlsx')
df_capacity.columns = ["h0", "h1", "h2", "h3", "h4", "h5"]
sum_of_capacity = df_capacity.sum()

for i in range(0, len(df_patients["p"])):
    x1 = int(int(df_patients["p"][i])) #COVID-19 patient count
    b1 = df_capacity["h1"] / sum_of_capacity #ratio of hospital capacity
    c = random.randint(0, 80) #transferred patients

    # y = b1 * x_1 + 0.5 * x_0 + c
    y = b1 * x1 + 0.5 * df_capacity["h1"] + c
    mymodel = numpy.poly1d(numpy.polyfit(x1, y, 3))
```

Please, refer to Section 8.1 to understand further how the dataset is utilized in our case study for hospital bed availability.

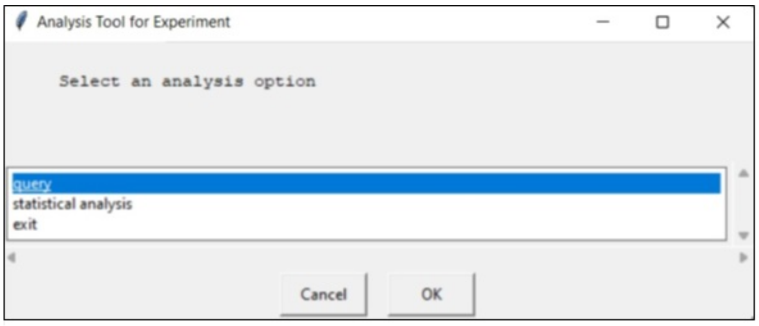
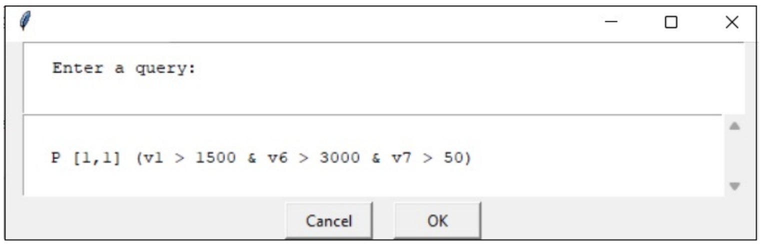


Figure 7.1: Analysis Tool user interface



```
P [1,1] (v1 > 1500 & v6 > 3000 & v7 > 50)
```

Figure 7.2: An example formula to execute for analysis

7.1.0.2 Hypothesis specification

The user specifies a hypothesis for a system under study (e.g., hospital bed availability prediction), in a formally defined hypothesis language (e.g., STL specification in Section 3.4.1). In order to define and test the STL formula, the user can get benefit from our *Analysis Tool* provided via our Kepler workflow. The tool accepts STL formula and is able to execute the formula on the generated dataset.

The interface for the *Analysis Tool* is given in Figure 7.1. The user should select the *query* option to start writing their formula.

As an example, we will try to analyze the formula in Figure 7.2 with the generated dataset. According to the formula, time traces that their previous time traces have *variable 1* bigger than *1500*, *variable 6* bigger than *3000* and *variable 7* bigger than *50* are found.

$$P[1,1](v1 > 1500 \& v6 > 3000 \& v7 > 50)$$

When the formula is executed, a graphic is depicted as an output. The output of the

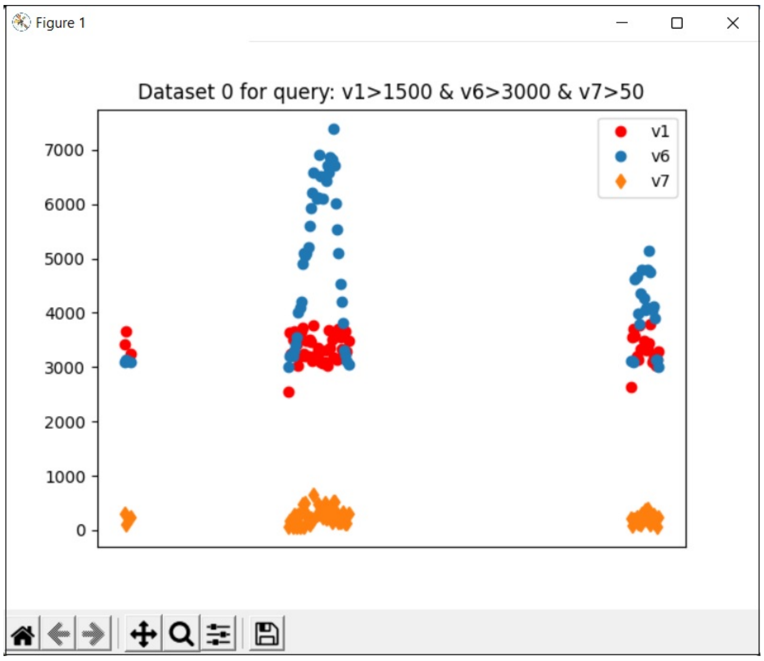


Figure 7.3: The output of the formula

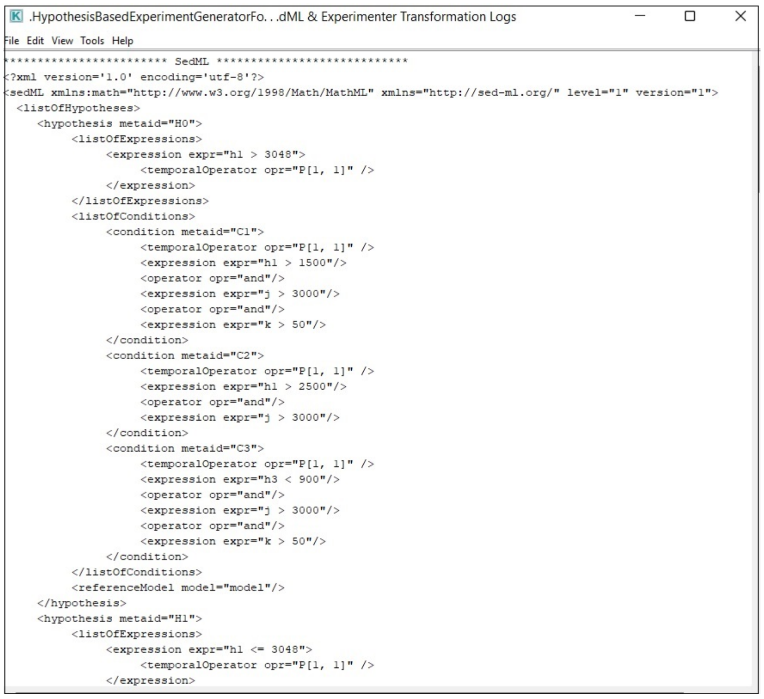
example formula is given in Figure 7.3. According to the graph, the values of the variables *v1, v6, and v7* are depicted on the time trace that they reside.

7.1.0.3 System specification

The user should provide a system specification where factors of the experiment, signals in the ptSTL formula, limit of the factors and the null hypothesis are defined. The null hypothesis is not mandatory. However, if it is given by the user, then the hypothesis that the user specified is considered as null hypothesis.

The user specification is formatted as followings:

1. An integer array for the number of factors of the experiments (e.g.,*[0, 1, 2, 3, 4, 5, 6, 7]*),
2. An integer array for the signals (e.g., *[6, 7]*),
3. A map for the limits of the factors (e.g., *'0': [60, 117]*),
4. A formula for an alternative hypothesis (e.g., $h1 < 3048$)


.HypothesisBasedExperimentGeneratorFo. . .dML & Experimenter Transformation Logs
File Edit View Tools Help

```
*********************** SedML ***************************
<?xml version='1.0' encoding='utf-8'?>
<sedML xmlns:math="http://www.w3.org/1998/Math/MathML" xmlns="http://sed-ml.org/" level="1" version="1">
  <listOfHypotheses>
     <hypothesis metaid="H0">
          <listOfExpressions>
               <expression expr="h1 > 3048">
                    <temporalOperator opr="P[1, 1]" />
               </expression>
          </listOfExpressions>
          <listOfConditions>
               <condition metaid="C1">
                    <temporalOperator opr="P[1, 1]" />
                    <expression expr="h1 > 1500"/>
                    <operator opr="and"/>
                    <expression expr="j > 3000"/>
                    <operator opr="and"/>
                    <expression expr="k > 50"/>
               </condition>
               <condition metaid="C2">
                    <temporalOperator opr="P[1, 1]" />
                    <expression expr="h1 > 2500"/>
                    <operator opr="and"/>
                    <expression expr="j > 3000"/>
               </condition>
               <condition metaid="C3">
                    <temporalOperator opr="P[1, 1]" />
                    <expression expr="h3 < 900"/>
                    <operator opr="and"/>
                    <expression expr="j > 3000"/>
                    <operator opr="and"/>
                    <expression expr="k > 50"/>
               </condition>
          </listOfConditions>
          <referenceModel model="model"/>
     </hypothesis>
     <hypothesis metaid="H1">
          <listOfExpressions>
               <expression expr="h1 <= 3048">
                    <temporalOperator opr="P[1, 1]" />
               </expression>
```

Figure 7.4: The output of experiment execution showing partial SED-ML content

Please, refer to Section 8.1 to understand how the dataset is utilized in our case study for hospital bed availability.

7.2 Execution & Post-execution of the Workflow

When the user completes the input preparation, then the workflow takes place. Firstly, experiment models are generated as files (e.g., XML for SED-ML and XPR for Xperimenter) and also shown as an output to the user by Kepler workflow in Figure 7.4. Similarly, the output of the experiment is also shown as an output where the count of hypothesis supporting and non-supporting time traces are listed including those time traces. An example output of an experiment result can be found in Figure 8.2. Afterwards, the *Analysis Tool* aids to further analyze the result, re-evaluate the hypothesis or the STL conditions.

Please, refer to Sections 8.1.2 and 8.3 to understand how the workflow behaves in our case study for hospital bed availability.

CHAPTER 8

CASE STUDY 1: A PREDICTIVE ANALYSIS OF HOSPITAL BED AVAILABILITY DURING COVID-19 PANDEMIC

In order to gain a deeper understanding of a megamodel and the process for the hypothesis-driven experiment design [2], we opted for starting with a current study domain to degrade the complexity of the proposed system. We consider that a hospital bed availability prediction system serves excellent for this purpose based on its importance, especially during COVID-19 pandemic. A rise in the number of COVID-19 patients burdens hospitals and it is also a valid indicator of the necessity of taking further measures against the pandemic.

8.1 Hospital Bed Availability

The Hospital Bed Availability study is modeled based on the hospitals in the capital city of Turkey, Ankara dedicated to serving the COVID-19 patients in Figure 8.1. There exist 6 hospitals and these hospitals are intentionally selected as they serve the most of the patients in Ankara. Authorities state that depending on the daily situation and their capacity, the hospitals transfer patients to the closest hospitals. For example, if the *Bilkent Sehir Hastanesi* gets filled up, the closest hospitals *Ankara Gazi Universitesi Hastanesi* and *Sehit Sait Erturk Devlet Hastanesi* will start to admit more patients than average, depending on the increase of daily COVID-19 patients. Thus, keeping the accurate number of occupancy and predicting the possible increase in the number of patients becomes quite important for the healthcare professionals.

As we propose to study predictive analysis on hospital bed availability, we stipulated two main required data: first is the bed capacity of each hospital, second is the

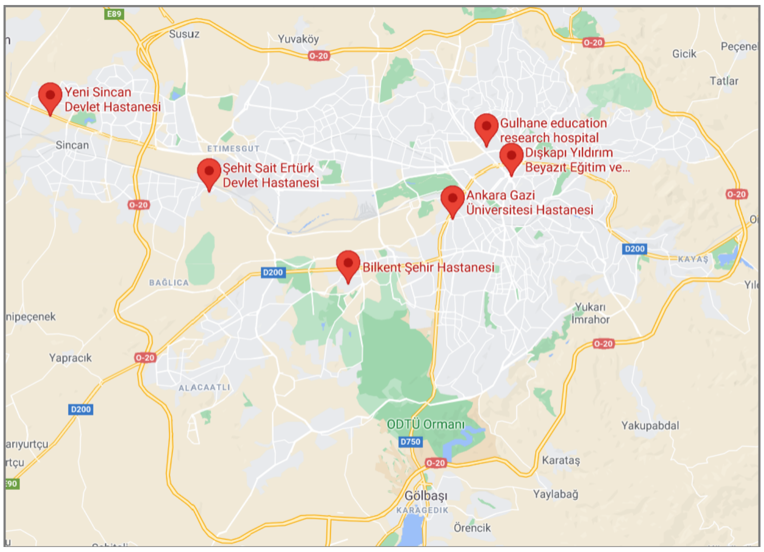

Yeni Sincan Devlet Hastanesi
Şehit Sait Ertürk Devlet Hastanesi
Gulhane education research hospital
Dışkapı Yıldırım Beyazıt Eğitim ve...
Ankara Gazi Üniversitesi Hastanesi
Bilkent Şehir Hastanesi
Sincan
Susuz
Yuvaköy
Gölbaşı
ODTÜ Ormanı
Yukarı İmrahor
Karataş
Yaylabağ
Yakupabdal
Örencik
Yapracık
Peçenek
Gicik
Tatlar

Figure 8.1: Selected hospitals with Covid-19 services in Ankara

daily number of hospitalized COVID-19 patients in Turkey. However, as of being the capital city of Turkey and its location, hospitals in Ankara also admit patients from other cities due to its hospital capacity. Therefore, a third parameter for the number of COVID-19 patients from neighbor cities are added to the study. Additionally, a state vector retains the number of daily bed occupancy of each hospital *i*, the number of daily COVID-19 patients *j*, and the daily number of admitted COVID-19 patients from the neighbor cities k at a certain state. We denote the daily number of bed occupancy h^i on each hospital *i*.

The capacity of the hospitals is given in Table 8.1 [55]. The capacity of the hospitals is *117, 3810, 300, 1150, 115,* and *480*, respectively. Considering the capacity of the *Bilkent Sehir Hastanesi*, the hospital becomes the major center where it should raise an alert in case of fullness. Finally, a hospital is considered to be over capacity by having many patients over 80% of the capacity.

8.1.1 Pre-execution: System Specification and Data Collection

The identified hospital bed capacity system owns several specific features and constraints (e.g., the number of daily bed occupancy of each hospital and the overall capacity of the hospitals) defining the self and creating the recognized problem. Ac-

Table 8.1: Selected hospitals with COVID-19 services in Ankara and their capacities

Number	**Hospital Name**	**Bed Capacity**
h0	Ankara Gazi Universitesi Hastanesi	117
h1	Bilkent Sehir Hastanesi	3810
h2	Diskapi Yildirim Beyazit Egitim ve Arastirma Hastanesi	300
h3	Gulhane education and research hospital	1150
h4	Sehit Sait Erturk Devlet Hastanesi	115
h5	Yeni Sincan Devlet Hastanesi	480

cordingly, those sets of specifications can be beneficial to introduce the system under investigation to the hypothesis-based experiment design workflow. The followings describe the fundamental specifications for the system under study. The capacity of the variables *j* and *k* were determined based on the total number of selected hospital capacities in Ankara multiplied by 10. The multiplication coefficient 10 represents the percentage of the daily number of hospitalized COVID-19 patients in Turkey, i.e., a maximum of *10%* [56].

1. An integer array for the number of daily bed occupancy of each hospital, number of hospitalized COVID-19 patients in Turkey and number of admitted COVID-19 patients from neighbor cities counts: *[0, 1, 2, 3, 4, 5, 6, 7]*,

2. An integer array for the non-capacity factors that are numbers representing the number of hospitalized COVID-19 patients in Turkey and number of admitted COVID-19 patients from neighbor cities counts: *[6, 7]*,

3. A map for all the hospitals with their capacity: *'h0': [60, 117], 'h1': [1905, 3810], 'h2': [150, 300], 'h3': [575, 1150], 'h4': [57, 115], 'h5': [240, 480], 'j': ['0', '59720'], 'k': ['0', '59720']*,

4. A formula to trace the non-supporting time traces of the hospitals calculated with the multiplication of its capacity and the capacity fullness ratio, i.e., 80% (e.g., $h1 < 3048$)

Unfortunately, we found the acquisition of authentic test data difficult as they are not shared per city by the Turkish authorities. This impediment motivated us toward data generation alternatives for the prevalent problem domain, i.e., hospital bed availability during COVID-19. Therefore, for the purpose of this study, a data generation algorithm using linear regression was employed to create data sets for the hospital bed availability in Ankara during COVID-19. The algorithm generates data for the daily number of the occupied beds of each hospital and the transferred number of patients from the neighboring cities by using the COVID-19 numbers of Turkey shared by the Republic of Turkey Ministry of Health [57], i.e., *number of patients for today*. We assumed that the hospitals had half of their capacity was already occupied by non-COVID-19 patients, when the pandemic has started. The regression model used for data generation is

$$Y = b_1X_1 + 0.5X_0 + c$$

where X_1 is the *number of patients for today* multiplied by *1/3* that represents the share of COVID-19 of the city of Ankara, b_1 is the ratio of the hospital capacity to total capacity of the all hospitals in Ankara and X_0 is the capacity of the hospital. Finally, *c* represents the random number which is limited to the daily patients from neighbor cities.

The algorithm essentially generates data sets linearly from an initial random number where the daily number of COVID-19 numbers of Turkey is an coefficient for a provided number of time traces. The data contains the temporal operator P (previously) and the time interval [0, 101]. Although valuable for generating lots of data, this algorithm has the disadvantage of generating relatively small non-supporting data.

The decisions about data generation made upon a requirement that the overall time traces for a single hospital avoiding the fullness should be as approximate as possible to the hospital's fullness measure, i.e., *80%*. As a result, we obtained 4 different data sets, and each data set contains *101* sequential time traces for every hospital and non-capacity variables that are number of hospitalized COVID-19 patients in Turkey and number of admitted COVID-19 patients from neighbor cities. We eventually achieved *404* time traces representing the last *14* months of COVID-19, where *62* of the time traces have hospital *h1* with over *80%* capacity, i.e., $h1 > 3048$.

8.1.2 Pre-execution: Hypotheses about Hospital Bed Availability in Ankara during COVID-19

Hospital bed availability became one of the major concerns in many countries during the COVID-19 pandemic. Discovering the conditions causing this fatal problem, at least in Ankara, before it aggravates is the concern of this case study. Thus, we formulated our concern based on the previously established steps of the scientific process in the Introduction section, with an appropriate question addressing the problem and hypotheses targeting to solve the problem. Specifically, the formalized questions as ptSTL formulas describe the hypotheses.

1. **Question:** What are the conditions that originate fulness on hospital *h1* on the next day?

2. **Conditions:** The following conditions, defined according to the formal specification in Section 3.4.1 originate fullness on hospital h1 on the next day:

$$\phi = \phi_1 \vee \phi_2 \vee \phi_3$$

$$\phi_1 = P_{[1,1]}((h1 > 1500) \wedge (j > 3000) \wedge (k > 50))$$

$$\phi_2 = P_{[1,1]}((h1 > 2500) \wedge (j > 3000))$$

$$\phi_3 = P_{[1,1]}((h3 > 900) \wedge (j > 3000) \wedge (k > 50))$$

   Each sub-formula ϕ_1, ϕ_2 and ϕ_3 states a condition that leads to fullness on hospital *h1* on the next day.

   - ϕ_1 : on the occasion of more than *1500* patients at hospital *h1*, the number of hospitalized COVID-19 patients is more than *3000*, and more than *50* patients get transfered to Ankara,
   - ϕ_2 : on the occasion of more than *2500* patients at hospital *h1* and the number of hospitalized COVID-19 patients is more than *3000*,
   - ϕ_3 : on the occasion of more than *900* patients at hospital *h3*, the number of hospitalized COVID-19 patients is more than *3000*, and more than *50* patients get transfered to Ankara.

3. **Null hypothesis (H0):** If one of the condition occurs, then the hospital *h1* observes fullness by growing *80%* over its capacity where the condition is

$$h1 > 3048.$$

4. **Alternative hypothesis (H1):** If one of the condition occurs, then the hospital *h1* does not observe fullness by growing *80%* over its capacity where the condition is

$$h1 <= 3048.$$

We assign our individual hypothesis-based experiment design workflow in Figure 6.2 for the remainder of the steps (4, 5, 6, and 7) of the scientific process. In the following sections, we explain how the workflow supervises the complete list of experiment procedures sequentially; specifically, design, execution, validation, and analysis.

8.2 Execution: Hypothesis to Experiment Model Transformations

Following the fulfillment of the user operations for the system under study, *Hypothesis 2 Experiment Transformator* module, i.e., the primary step in Figure 6.2, initiates the simulation experiment workflow. Having the system specifications and the hypotheses is the compulsory provision to employ the tasks for SED-ML model generation from system specification and from SED-ML to Xperimenter model transformation. It is pertinent to remark that generated datasets are only necessary for the later phases of the workflow, e.g., experiment execution.

The module, an individualized Python script, is solely liable for the experiment model obtaining in two ways: model generation and model transformation. For this study, we underline how we interpret these two similar tasks: while we describe the model transformations as a practice over two or more conventional models serving the same domain, e.g., DSLs, we contemplate the data generation as another practice between any custom specification. In light of this, *Hypothesis 2 Experiment Transformator* practices the following functions:

1. From user-defined system specification to SED-ML model generation,

2. From SED-ML to Xperimenter model transformation.

The following sections explain the proposed hypothesis extension to SED-ML alongside the model generation, and SED-ML to Xperimenter model transformations, sequentially.

8.2.1 Hypothesis Extension to SED-ML

We propose a hypothesis extension to SED-ML to attain a solution for the lacking association issue between an experiment and its hypothesis. The proposed SED-ML model gracefully interprets the STL semantics into a markup language, i.e., XML.

The followings clarifies how the user-defined system specification and the hypotheses describe the SED-ML model accordingly with Table 8.2. We set the default initial values for the SED-ML model generation with the fact that one experiment associated with a single task and an experiment model can sufficiently prove or refute a list of hypotheses enclosed to a question.

1. *An integer array for the number of daily bed occupancy of each hospital, the number of hospitalized COVID-19 patients, and admitted COVID-19 patients from the neighbor cities:* transformed into variables in the data generator of a task,
2. *A map for hospital capacities:* transformed into variable limits,
3. *Hypotheses:* transformed into list of hypotheses.

Table 8.2: User-defined specifications to SED-ML Mapping

No	**System specifications**	**SED-ML**
1	hypotheses	listOfHypotheses
2	default initial values for a single simulation	listOfSimulations
3	default initial values for a single model	listOfModels
4	default initial values for a single task	listOfTasks
5	system specification	listOfDataGenerators

We presented the extended *listOfHypotheses* for a single variable part of the generated SED-ML model in the Listing 8.2.1. A hypothesis in SED-ML consists of an expression that defines the hypothesis, itself, and three conditions with multiple expressions, and the expression relations are defined with *and* for this specific example. The *temporalOperators* are *P[1, 1]* for each condition and expression. And, the *relation* entity is used to explain the hypothesis *H0* with its conditions for supporting or non-supporting, as there is a single hypothesis. It is important to note that *H1* uses the same conditions with *H0*, as the relation between the hypotheses is *CONTRADICT*. Finally, *H0* has a *referenceModel* that relates the expressions to an actual model.

```xml
<listOfHypotheses>
        <hypothesis metaid="H0">
                <listOfExpressions>
                        <expression expr="h1 > 3048">
                                <temporalOperator opr="P[1, 1]" />
                        </expression>
              </listOfExpressions>
                <listOfConditions>
                        <condition metaid="C1">
                                <temporalOperator opr="P[1, 1]" />
                                <expression expr="h1 > 1500"/>
                                <operator opr="and"/>
                                <expression expr="j > 3000"/>
                                <operator opr="and"/>
                                <expression expr="k > 50"/>
                        </condition>
                        <condition metaid="C2">
                                <temporalOperator opr="P[1, 1]" />
                                <expression expr="h1 > 2500"/>
                                <operator opr="and"/>
                                <expression expr="j > 3000"/>
                        </condition>
                        <condition metaid="C3">
                                <temporalOperator opr="P[1, 1]" />
                                <expression expr="h3 < 900"/>
                                <operator opr="and"/>
                                <expression expr="j > 3000"/>
                                <operator opr="and"/>
                                <expression expr="k > 50"/>
                        </condition>
                </listOfConditions>
                <referenceModel model="model"/>
        </hypothesis>
        <hypothesis metaid="H1">
                <listOfExpressions>
                        <expression expr="h1 <= 3048">
                                <temporalOperator opr="P[1, 1]" />
                        </expression>
              </listOfExpressions>
        </hypothesis>
        <listOfRelations>
                <relation relation="CONTRADICT">
                        <hypothesis hyp="H0"/>
                        <hypothesis hyp="H1"/>
                </relation>
        </listOfRelations>
</listOfHypotheses>
<listOfModels>
   <model metaid="model" source="/sourceModel" />
</listOfModels>

```

8.2.2 Execution: SED-ML to Xperimenter Model Transformation

The generation of another experiment model alongside the SED-ML is an essential effort to enrich the megamodel for the experimenters. The intention supporting this effort is to encourage the experimenters to develop their DSLs serving their particular needs, introduce them to the megamodel, and find common ground to bestow the knowledge. With this in mind, we undertook the Xperimenter model transformation from SED-ML, and achieved the Xperimenter model in the Listing 8.2.2.

We opted to apply model transformation from SED-ML to Xperimenter rather than the STL to Xperimenter. Because the SED-ML specification represents a formal model for capturing the essentials of simulation experiments including hypotheses and this method improves the ability to create traceability to the lower models, i.e., Xperimenter.

It is crucial to note that the STL formula defining the hypotheses was given as user input. In order to keep the originality of the Xperimenter model and as it was not a primary goal in this research, those inputs were not translated into Xperimenter and processed by the Python script in Appendix D.

```
experiment hospitalCapacity
desc "predictive analysis on hospital bed availability";
objective COMPARATIVE;
design hospitalCapacityDesign;
simulation hospitalCapacitySimulation;
visual DEFAULT;
target KEPLER;

variable h0: INTEGER group FACTOR [0, 117];
variable h1: INTEGER group FACTOR [0, 3810];
variable h2: INTEGER group FACTOR [0, 300];
variable h3: INTEGER group FACTOR [0, 1150];
variable h4: INTEGER group FACTOR [0, 115];
variable h5: INTEGER group FACTOR [0, 480];
variable j: INTEGER group FACTOR [0, 59720];
variable k: INTEGER group FACTOR [0, 59720];
design hospitalCapacityDesign
method FULLFACTORIAL;
varlist h0 h1 h2 h3 h4 h5 j k ;

simulation hospitalCapacitySimulation
modelFile /hospitalData/;
modelType DISCRETEEVENT;
inport h0: h0;
inport h1: h1;
inport h2: h2;
inport h3: h3;
inport h4: h4;
inport h5: h5;
inport j: j;
inport k: k;

```

Listing 8.1: Xperimenter model for the hospital experiment

The model transformation from SED-ML to Xperimenter is a relatively straightforward duty as both of the SED-ML and Xperimenter models possess many mutual variables. Table 8.3 summarizes the variable mapping effort from SED-ML to Xperimenter. An Xperimenter model starts with an *experiment* specification containing a *description*, an *objective*, a *design*, a *simulation*, an *analysis*, a *visual* and a *tar-*

get information. The *model* and *simulation* variables from SED-ML is equivalent to the same variables in Xperimenter. Having said that, while the *task* is representing the *experiment*, the *dataGenerator* and *output* collectively represents *variable* in Xperimenter. And, *variable* of the *task* are transformed into *varList* of *design* for Xperimenter.

Table 8.3: SED-ML to Xperimenter Variable Mapping

No	**SED-ML**	**Xperimenter**
1	model	model
2	simulation	simulation
3	task	experiment
4	dataGenerator	variable
5	output	variable
6	task.variable	design.varList

8.3 Experiment Execution

Once achieving the experiment models, the execution phase of the experimentation process inaugurates the workflow for the execution. The experiment execution module, i.e., the second step in Figure 6.2, exclusively consists of a Python script that takes the generated data sets, the user-defined system specification, and the previously generated SED-ML model as inputs and determines the time traces that support and non-support the hypotheses. The script is fundamentally responsible for the following tasks:

1. Interpreting experiment model, i.e., SED-ML specification, to collect the hypotheses,
2. Interpreting the data set on the basis of the system specifications,
3. Executing the conditions against the data set to find the hypothesis supporting and non-supporting time traces.

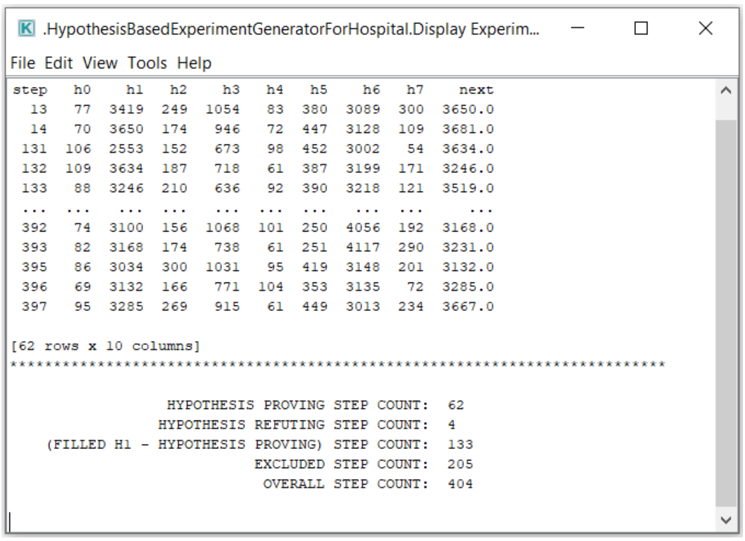
The experiment run accumulates throughputs for the number of successful and failing conditions, the overall number of time traces, the number of skipped data, and finally prints out the results in Figure 8.2. The screenshot precisely contains the following information:

1. The number of time traces that successfully support the conditions,
2. The number of the filled *h1* traces where previous traces non-supporting the conditions,
3. The number of time traces that are non-supporting the conditions where the next trace is not over the capacity ($h1 < 3048$),
4. The overall number of skipped time traces due to the non-supporting conditions for the hypotheses,
5. The overall number of traces that the experiment used, excluding the first ten time traces in each dataset to enhance the quality of the input by eliminating the initial randomized time traces.

As a result, the number of time traces that successfully support the conditions *(62 out of 66)* is sufficient to claim that the hypothesis is proven with a margin less than *%10*.

8.4 Post-Execution: Experiment Validation

Trace analysis is a useful technique for verifying formal proofs. A trace checker analyses the traces and outlines any violations of the proffered formula. Due to its frugality and practicality of the method, employing a trace checker for STL specifications appears to be reasonable in terms of experiment result validation in this study, even though formal proofing is not in the scope of this dissertation. Taking that into consideration, we employed the STL Trace Checker [3] to validate the experiment output that we formerly conducted. The trace checker takes the previously stated conditions for the hospital bed availability analysis for the hospital *h1* alongside the generated datasets and returns the supporting and non-supporting data traces. The output of the STL Trace Checker appears to tally with our expectations for the number of the traces


.HypothesisBasedExperimentGeneratorForHospital.Display Experim...
File Edit View Tools Help

```
step   h0    h1   h2    h3   h4   h5    h6   h7    next
  13   77  3419  249  1054   83  380  3089  300  3650.0
  14   70  3650  174   946   72  447  3128  109  3681.0
 131  106  2553  152   673   98  452  3002   54  3634.0
 132  109  3634  187   718   61  387  3199  171  3246.0
 133   88  3246  210   636   92  390  3218  121  3519.0
 ...  ...   ...  ...   ...  ...  ...   ...  ...     ...
 392   74  3100  156  1068  101  250  4056  192  3168.0
 393   82  3168  174   738   61  251  4117  290  3231.0
 395   86  3034  300  1031   95  419  3148  201  3132.0
 396   69  3132  166   771  104  353  3135   72  3285.0
 397   95  3285  269   915   61  449  3013  234  3667.0

[62 rows x 10 columns]
***************************************************************************

                 HYPOTHESIS PROVING STEP COUNT:  62
                HYPOTHESIS REFUTING STEP COUNT:  4
    (FILLED H1 - HYPOTHESIS PROVING) STEP COUNT:  133
                           EXCLUDED STEP COUNT:  205
                            OVERALL STEP COUNT:  404
```

Figure 8.2: Hospital bed availability experiment result including the time traces with the hospital capacities that supporting the hypothesis

supporting the conditions, i.e., *62*, and the number of the traces the non-supporting the conditions, i.e., *4*.

Based on the quantitative comparison of the throughputs from the STL Trace Checker and the experiment run we conducted, we also observed that the throughputs are consistent with the analysis reported by us. The compatibility of the analysis results demonstrates the adequacy of using a trace checker for the validation of formally specified hypotheses.

8.5 Post-Execution: Experiment Analysis

The final phase of the scientific experimentation process is to evaluate the acquired throughputs with the help of prevalent analytical methods. These analytical techniques assist in collecting and modeling data in the process of decision making. We, hence, offer an STL based experiment statistical analysis software, embedded in the workflow as the final step in the Figure 8.1, that helps the experiment designers in their endeavour of data analysis. The proposed statistical analysis tool utilizes the statistical capabilities of the Python programming language.

The tool is capable of applying the following statistical methods on a given dataset:

1. Supporting or non-supporting an STL formula on a dataset,
2. Applying the following statistical analysis:
    (a) Histogram of data sets
    (b) Linear regression
    (c) Statistical summary

We opted for a humble statistical search to scrutinize the utilized datasets in terms of the quality aspect, and for the illustration purposes of the offered analytical tool. To attain this objective, we revisit and expand the formerly exploited hypotheses specification with a condition where the hospital *h1* has a number of patients less than *80%* of its capacity in the next trace.

$$\phi = (\phi_1 \vee \phi_2 \vee \phi_3) \wedge \phi_4$$

$$\phi_4 = P_{[1,1]}(h1_{next} < 3048)$$

The execution of the analysis produces graphical throughput in Figure 8.3. The throughput highlights that in every five trace intervals for the aggregated dataset, there exist two or more traces that non-supporting the hypothesis. Bear in mind that the graphic depicts the aggregation of the 4 different datasets where each dataset contains 101 traces. Explained differently, approximately *1* traces in *101* traces non-supporting the hypothesis, and the analysis outcome is considerably close to our experiment result for the non-supporting cases. The significance of this analysis for the data quality can be assumed marginal regarding the aim of the study, i.e., having a complete workflow for the simulation experiment.

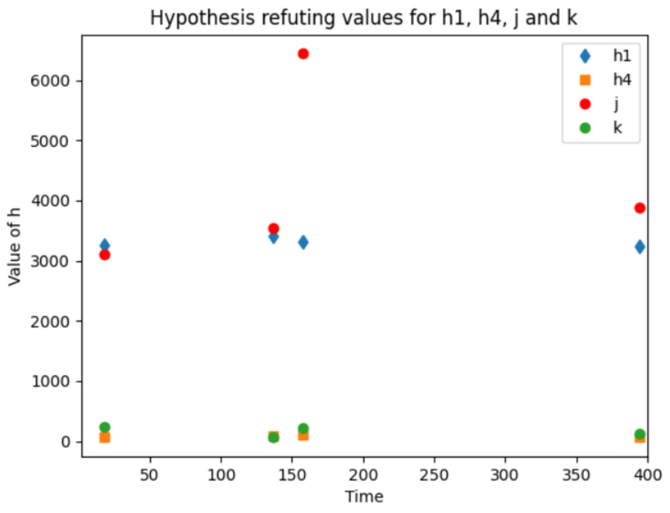


Figure 8.3: Time traces of the hospital capacities that non-supporting the hypothesis

CHAPTER 9

CASE STUDY 2: CONGESTION ANALYSIS ON A TRAFFIC NETWORK

We selected a simpler second case study to reduce the complexity of the proposed system. In that sense, we believe that a traffic system is very appropriate for a variety of analysis and adjustability to other problem fields.

9.0.1 A Traffic Network

The traffic network under study that is modeled as a piecewise affine system [58] is illustrated in Figure 9.1. It consists of 6 links and 2 traffic signals. This network design is intentionally employed as it contains enough signals (at least two) to investigate congestion on a single link.

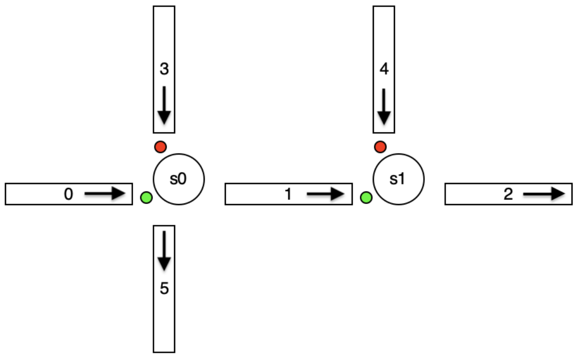


Figure 9.1: Traffic network containing 2 signals and 6 links [3]

As we propose to investigate congestion analysis on a traffic network, we stipulated two required data: first is the capacity of vehicles on each link, second is a set of traffic signal values. Additionally, a state vector retains the number of vehicles on each link i and the configuration of each signal j at a certain state. We denote the number of vehicles v^i on each link i, and the configuration s^j of each signal j.

The signals can be 0 or 1, where $s^i = 0$ and $s^i = 1$ stating that traffic is allowed in horizontal direction and vertical direction, respectively. The restraint ensures that the traffic only flows in either vertical or horizontal direction. The direction of a link determines the capacity of that link. With this in mind, the capacity of the links on the horizontal direction is set to 20, whereas the capacity of the links on the vertical direction is set to 40. In particular, the capacity of links 0, 1 and 2 are set to 40; and the capacity of links 3, 4 and 5 are set to 20, i.e., $v^i \in [0, 40]$ for $i \in 0, 1, 2$; and $v^i \in [0, 20]$ for $i \in 3, 4, 5$. Finally, the congestion on a link is measured by having many vehicles over 75% of the capacity.

9.0.1.1 Pre-execution: System Specification and Data Collection

The identified traffic network system owns several specific features and constraints (e.g., the number of links and signals belong to the traffic network, and the capacity of the links) defining the self and creating the recognized congestion problem. Accordingly, those set of specifications can be beneficial to introduce the system under investigation to the hypothesis-based experiment design workflow. The followings describe the fundamental specifications for a traffic network.

1. An integer array for all the link and signal counts: *[0, 1, 2, 3, 4, 5, 6, 7]*,
2. An integer array for the numbers representing the signals: *[6, 7]*,
3. A map for all the links with their capacity: *'v2': [0, 40.0], 'v3': [0, 20.0], 'v0': [0, 40.0], 'v1': [0, 40.0], 'v6': ['a', 'b'], 'v7': ['a', 'b'], 'v4': [0, 20.0], 'v5': [0, 20.0]*,
4. A formula to trace the non-supporting time traces of each link calculated with the multiplication of its capacity and the congestion ratio, i.e., 75% (e.g., $v_1 < 30$)

A data generation algorithm [3] that simulates a system under study from random initial states was employed to create data sets for the traffic network for the purpose of this study. The algorithm essentially builds data sets from random initial conditions for a provided number of time traces. Although valuable for generating lots of data, this algorithm has the disadvantage of generating relatively small non-supporting data. Nevertheless, dismissing up to 10% of the first random data samples solved the inconvenience straightforwardly.

The decisions about data generation made upon a requirement that the overall time traces for a single link avoiding the congestion should be as approximate as possible to the link's congestion measure, i.e., 75%. As a result, we obtained 20 different data sets, and each data set contains 100 sequential time traces for every link and signal on the traffic network. We eventually achieved 2000 time traces, where 456 of the time traces have link v_1 with over 75% capacity, i.e., $v^1 > 30$.

9.0.1.2 Pre-execution: Hypotheses about Traffic Congestion

Traffic congestion typically occurs due to many vehicles, when a volume of traffic generates demand for space longer than the available link capacity. Discovering the conditions causing this real-life problem before it aggravates is the concern of this case study. Thus, we formulated our concern based on the previously established steps of the scientific process in the Introduction section, with an appropriate question addressing the problem and hypotheses targeting to solve the problem. Specifically, the formalized questions as ptSTL formulas describe the hypotheses.

1. **Question:** What are the conditions that originate congestion on link v_1 in the next time state?
2. **Conditions:** The following conditions, defined as formal specification [3] originate congestion on link v_1 in the next time state:

$$\phi = \phi_1 \vee \phi_2 \vee \phi_3$$

$$\phi_1 = P_{[1,1]}((v^1 > 15) \wedge (s^1 > 1) \wedge (s^0 > 0))$$

$$\phi_2 = P_{[1,1]}((v^1 > 25) \wedge (s^1 > 1))$$

$$\phi_3 = P_{[1,1]}((v^4 < 10) \wedge (s^1 > 1) \wedge (s^0 > 0))$$

Each sub-formula ϕ_1, ϕ_2 and ϕ_3 states a condition that leads to congestion on link v_1 in the next time trace.

- ϕ_1 : on the occasion of more than 15 vehicles on link v_1, if s_1 blocks link v_1 and s_0 allows flow of vehicles from link v_0 to link v_1,
- ϕ_2 : on the occasion of more than 25 vehicles on link v_1, if s_1 blocks link v_1,
- ϕ_3 : on the occasion of less than 10 vehicles on link s_4, if s_1 blocks link v_1 and s_0 allows flow of vehicles from link v_0 to link v_1.

3. **Null Hypothesis (H0):** If one of the condition occurs, then the link v_1 observes congestion by growing 75% over its capacity where the condition is

$$v^1 > 30.$$

4. **Alternative Hypothesis (H1):** If one of the condition occurs, then the link v_1 does not observe congestion by growing 75% over its capacity where the condition is

$$v^1 <= 30.$$

We assign our individual hypothesis-based experiment design workflow in Figure 6.2 for the remainder of the steps (4, 5, 6, and 7) of the scientific process. In the following sections, we explain how the workflow supervises the complete list of experiment procedures sequentially; specifically, design, execution, validation, and analysis.

9.0.2 Execution: Hypothesis to Experiment Model Transformations

After the user operations, *Hypothesis 2 Experiment Transformator* module, i.e., the primary step in Figure 6.2, initiates the simulation experiment workflow. Similar to previous case study, first the extended SED-ML model from hypothesis, then the Xperimenter model is generated.

9.0.2.1 Hypothesis Extension to SED-ML

Based on the Table 8.2, hypothesis to SED-ML model generation is applied. We, again, set the default values for the SED-ML model generation with the fact that one experiment associated with a single task and an experiment model can sufficiently prove or refute a list of hypotheses enclosed to a question.

The transformation mapping is given below:

1. *An integer array for all the link and signal counts:* transformed into variables in the data generator of a task,
2. *A map for link and signals with their capacities:* transformed into variable limits,
3. *Hypotheses:* transformed into list of hypotheses.

The generated SED-ML model is given in the Listing 9.0.2.1.

```
<listOfHypotheses>
                <hypothesis metaid="H0">
                        <expressions>
                                <expression expr="v1 < 30"/>
                              </expressions>
                        <conditions>
                                <condition metaid="C1">
                                        <expression expr="v1 > 15"/>
                                        <operator opr="and"/>
                                        <expression expr="s1 > 1"/>
                                        <operator opr="and"/>
                                        <expression expr="s0 > 0"/>
                                </condition>
                                 <condition metaid="C2">
                                        <expression expr="v1 > 25"/>
                                        <operator opr="and"/>
                                        <expression expr="s1 > 1"/>
                                </condition>
                                 <condition metaid="C2">
                                        <expression expr="v4 < 10"/>
                                        <operator opr="and"/>
                                        <expression expr="s1 > 1"/>
                                        <operator opr="and"/>
                                        <expression expr="s0 > 0"/>
                                </condition>
                        </conditions>
                </hypothesis>
                <hypothesis metaid="H1">
                        <expressions>
                                <expression expr="v1 >= 30"/>
                              </expressions>
                </hypothesis>
                <coherenceLinks>
                        <coherenceLink coherence="EXPLAIN">
                                <hypothesis hyp="H0">
                                <relation rel="against"/>
                                <hypothesis hyp="H1">
                        </coherenceLink>
                </coherenceLinks>
        </listOfHypotheses>

         <listOfSimulations>
                    <uniformTimeCourse id="simulation">
                   </uniformTimeCourse>
          </listOfSimulations>

          <listOfModels>
                    <model id="model"/>
          </listOfModels>

          <listOfTasks>
                    <task simulationReference="simulation" modelReference="model"
                    ↪   id="task"/>
          </listOfTasks>

          <listOfDataGenerators>
                    <dataGenerator id="v0" name="v0">
                              <listOfVariables>
                                <variable id="v0" name="v0" taskReference="task" />
                              </listOfVariables>
                    </dataGenerator>
          </listOfDataGenerators>
```

9.0.2.2 SED-ML to Xperimenter Model Transformation

After achieving the SED-ML model, then Xperimenter model transformation from SED-ML is executed, and the Xperimenter model in the Listing 9.0.2.2 is generated.

The model transformation from SED-ML to Xperimenter can be revisited via the Table 8.3.

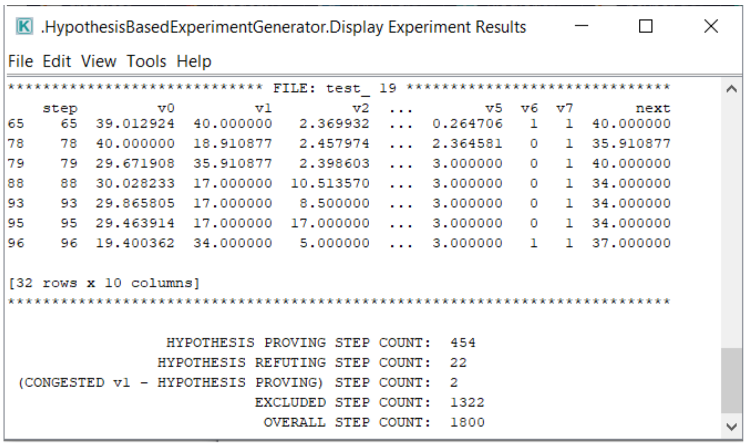
```
experiment network
desc "congestion analysis on a traffic network";
objective COMPARATIVE;
design networkDesign;
simulation networkSimulation;
analysis AnovaAnalysis;
visual DEFAULT;
target KEPLER;

variable v0: INTEGER group FACTOR [0, 40.0];
variable v1: INTEGER group FACTOR [0, 40.0];
variable v2: INTEGER group FACTOR [0, 40.0];
variable v3: INTEGER group FACTOR [0, 20.0];
variable v4: INTEGER group FACTOR [0, 20.0];
variable v5: INTEGER group FACTOR [0, 20.0];
variable s0: BOOLEAN group RESPONSE;
variable s1: BOOLEAN group RESPONSE;
design networkDesign
method FULLFACTORIAL;
varlist v0 v1 v2 v3 v4 v5 s0 s1 ;

simulation networkSimulation
modelFile /trafficdata16/;
modelType DISCRETEEVENT;
inport v0: v0;
inport v1: v1;
inport v2: v2;
inport v3: v3;
inport v4: v4;
inport v5: v5;
inport s0: s0;
inport s1: s1;

```

Listing 9.1: Xperimenter model for the traffic network experiment


.HypothesisBasedExperimentGenerator.Display Experiment Results
File Edit View Tools Help

```
**************************** FILE: test_ 19 *****************************
    step         v0         v1         v2  ...         v5  v6  v7       next
65    65  39.012924  40.000000   2.369932  ...   0.264706   1   1  40.000000
78    78  40.000000  18.910877   2.457974  ...   2.364581   0   1  35.910877
79    79  29.671908  35.910877   2.398603  ...   3.000000   0   1  40.000000
88    88  30.028233  17.000000  10.513570  ...   3.000000   0   1  34.000000
93    93  29.865805  17.000000   8.500000  ...   3.000000   0   1  34.000000
95    95  29.463914  17.000000  17.000000  ...   3.000000   0   1  34.000000
96    96  19.400362  34.000000   5.000000  ...   3.000000   1   1  37.000000

[32 rows x 10 columns]
**************************************************************************


                 HYPOTHESIS PROVING STEP COUNT:  454
                HYPOTHESIS REFUTING STEP COUNT:  22
 (CONGESTED v1 - HYPOTHESIS PROVING) STEP COUNT:  2
                           EXCLUDED STEP COUNT:  1322
                            OVERALL STEP COUNT:  1800
```

Figure 9.2: Traffic network experiment result

9.0.3 Experiment Execution

Once achieving the experiment models, the execution phase of the experimentation process in Figure 6.2 triggers the workflow for the execution. The output of the experiment run can be seen in Figure 9.2.

The screenshot precisely contains the following information:

1. The number of time traces that successfully support the conditions,
2. The number of the congested line v_1 traces where previous traces non-supporting the conditions,
3. The number of time traces that are non-supporting the conditions where the next trace is not over the capacity ($v^1 < 30$),
4. The overall number of skipped time traces due to the non-supporting conditions for the hypotheses,
5. The overall number of traces that the experiment used, excluding the first ten time traces in each dataset to enhance the quality of the input by eliminating the initial randomized time traces.

9.0.4 Post-execution: Experiment Validation

We employed the same STL Trace Checker [3] to validate the the second experiment output that we conducted. The trace checker takes the previously stated conditions for the congestion analysis on link v_1 alongside the generated datasets and returns the supporting and non-supporting data traces. The output of the STL Trace Checker appears to comply with our expectations for the number of the traces supporting the conditions, i.e., 454, and the number of the traces the non-supporting the conditions, i.e., 22.

Based on the quantitative comparison of the throughputs from the STL Trace Checker and the experiment run we conducted, we observed that the throughputs are consistent with the analysis reported by [3].

9.0.5 Post-execution: Experiment Analysis

To complete the process and to have an analytical prespective of the current case study, we revisit and expand the hypotheses specification with a condition where the link v_1 has a number of vehicles less than 75% of its capacity in the next trace.

$$\phi = (\phi_1 \vee \phi_2 \vee \phi_3) \wedge \phi_4$$

$$\phi_4 = P_{[1,1]}(v^1_{next} < 30)$$

The execution of the analysis produces graphical throughput in Figure 9.3. The throughput shows that in every five trace intervals for the aggregated dataset, there exist two or more traces that non-supporting the hypothesis. Bear in mind that the graphic depicts the aggregation of the 20 different datasets where each dataset contains 100 traces. In other words, approximately 20 traces in 2000 traces are non-supporting the hypothesis, and the analysis outcome is very much close to our experiment result for the non-supporting cases.

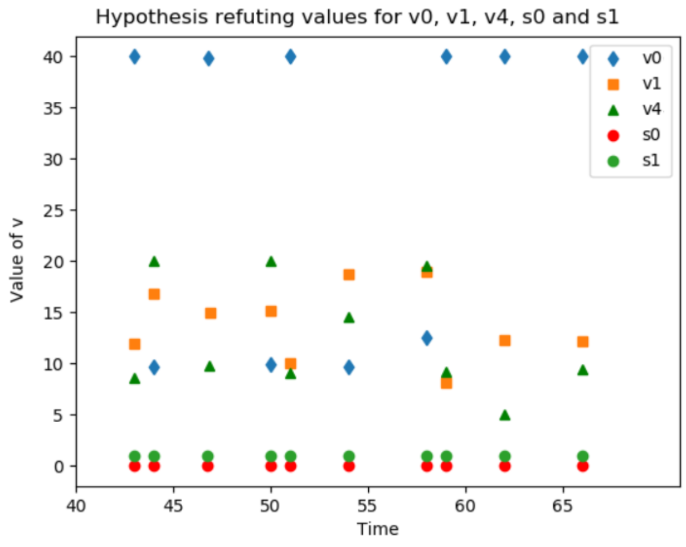


Figure 9.3: Throughput of the analysis

CHAPTER 10

CONCLUSION

Our research demonstrates that we took the first step in creating a reference implementation of a Global Model Management environment for Hypothesis-driven Experiment Design. We really profited from megamodeling approaches, which speed up the administration of models with a range of procedures, throughout this endeavour. In addition, we conducted a case study that bridges the gap between a theory and the relevant experiment by doing an experiment based on the hypothesis. Overall, our case study for hypothesis-based experiment design uses a well recognized simulation experiment description language, SED-ML, to combine experimental design, model, and hypotheses in a single process. With a scalable experimental medium, it improves the replicability of scientific experiments and the reproducibility of scientific discoveries. We believe that our research will serve as a base for future studies on the experiment design, such as expanding our experiment design with *Type I* and *II Errors* for complete statistical hypothesis testing. Furthermore, we realize that expanding our hypothesis definition with more statistical characteristics, such as significance level, is a worthy effort.

We realize that future investigations are needed to improve the precision and reliability of the specification mechanisms and introduce new model transformation algorithms that map hypotheses to experiment designs as we expand the megamodel with new DSLs and formalisms for experiment design and management systems. As a result of this foresight, we have decided to combine agent technology with our megamodel. The concept of combining cognitive computing with simulation experiments in a Model-driven Engineering (MDE) environment [7, 59] is to handle diverse tasks in scientific discovery such as describing challenges, constructing methods to model a system, and answering questions.

REFERENCES

[1] F. Bergmann, J. Cooper, M. König, I. Moraru, D. Nickerson, N. Novère, B. G Olivier, S. Sahle, L. Smith, and D. Waltemath, “Simulation experiment description markup language (sed-ml) level 1 version 3 (l1v3),” 03 2018.

[2] S. Çam, O. Dayıbaş, B. K. Görür, H. Oğuztüzün, L. Yilmaz, S. Chakladar, K. Doud, A. E. Smith, and A. Teran-Somohano, “Supporting simulation experiments with megamodeling,” in *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*, pp. 372–378, INSTICC, SciTePress, 2018.

[3] M. Ergürtuna and E. A. Göl, “An efficient formula synthesis method with past signal temporal logic,” *IFAC-PapersOnLine*, vol. 52, no. 11, pp. 43–48, 2019. 5th IFAC Conference on Intelligent Control and Automation Sciences ICONS 2019.

[4] D. Montgomery, *Design and Analysis of Experiments*. New Jersey, USA: John Wiley & Sons, Inc., 2006.

[5] Z. Merali, “Computational science: Error, why scientific programming does not compute,” *Nature*, vol. 467, pp. 775–777, Oct. 2010.

[6] L. N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. OHara, D. Gavaghan, and S. Emmott, “Troubling trends in scientific software use,” *Science*, vol. 340, no. 6134, pp. 814–815, 2013.

[7] L. Yilmaz, S. Chakladar, and K. Doud, “The goal-hypothesis-experiment framework: A generative cognitive domain architecture for simulation experiment management,” in *Proceedings of the 2016 Winter Simulation Conference (WSC)*, pp. 1001–1012, 2016.

[8] D. Köhn and N. Le Novère, “Sed-ml – an xml format for the implementation of the miase guidelines,” in *Computational Methods in Systems Biology* (M. Heiner

and A. M. Uhrmacher, eds.), (Berlin, Heidelberg), pp. 176–190, Springer Berlin Heidelberg, 2008.

[9] D. Waltemath, R. Adams, D. Beard, F. Bergmann, U. Bhalla, R. Britten, V. Chelliah, M. Cooling, J. Cooper, E. Crampin, A. Garny, S. Hoops, M. Hucka, P. Hunter, E. Klipp, C. Laibe, A. Miller, K. Moraru, D. Nickerson, P. Nielsen, M. Nikolski, S. Sahle, H. Sauro, H. Schmidt, J. Snoep, D. Tolle, O. Wolkenhauer, and N. le Novere, "Minimum information about a simulation experiment (miase).," *PLoS Computational Biology*, vol. 7, no. 4, 2011.

[10] J. Bèzivin, F. Jouault, and P. Valduriez, "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*, 2004.

[11] R. D. King, J. Rowland, S. G. Oliver, M. Young, W. Aubrey, E. Byrne, M. Liakata, M. Markham, P. Pir, L. N. Soldatova, A. Sparkes, K. E. Whelan, and A. Clare, "The automation of science," *Science*, vol. 324, no. 5923, pp. 85–89, 2009.

[12] D. Waltz and B. G. Buchanan, "Automating science," *Science*, vol. 324, no. 5923, pp. 43–44, 2009.

[13] H. Rahmandad and J. D. Sterman, "Reporting guidelines for simulation based research in social sciences," *System Dynamics Review*, vol. 28, no. 4, pp. 396–411, 2012.

[14] A. Ruscheinski, T. Warnke, and A. M. Uhrmacher, "Artifact-based workflows for supporting simulation studies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 6, pp. 1064–1078, 2020.

[15] P. Wilsdorf, M. Dombrowsky, A. M. Uhrmacher, J. Zimmermann, and U. v. Rienen, "Simulation experiment schemas –beyond tools and simulation approaches," in *2019 Winter Simulation Conference (WSC)*, pp. 2783–2794, 2019.

[16] F. Lorig, D. S. Lebherz, J. O. Berndt, and I. J. Timm, "Hypothesis-driven experiment design in computer simulation studies," in *2017 Winter Simulation Conference (WSC)*, pp. 1360–1371, Dec 2017.

[17] S. Chakladar, “A model driven engineering framework for simulation experiment management,” Master’s thesis, Auburn University, Alabama, USA, 2016.

[18] C. W. Krueger, “Software reuse,” *ACM Computing Surveys*, vol. 24, p. 131–183, jun 1992.

[19] R. Ewald and A. M. Uhrmacher, “Sessl: A domain-specific language for simulation experiments,” *ACM Transactions on Modeling and Computer Simulation*, vol. 24, pp. 11:1–11:25, Feb. 2014.

[20] A. Hallagan, B. Ward, and L. F. Perrone, “An experiment automation framework for ns-3,” in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools ’10, (ICST, Brussels, Belgium), pp. 38:1–38:2, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[21] L. F. Perrone, C. S. Main, and B. C. Ward, “Safe: Simulation automation framework for experiments,” in *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pp. 1–12, 2012.

[22] T. Warnke and A. M. Uhrmacher, “Reproducible parallel simulation experiments via pure functional programming,” in *Proceedings of the 2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–8, 2019.

[23] G. Blondet, J. L. Duigou, N. Boudaoud, and B. Eynard, “An ontology for numerical design of experiments processes,” *Computers in Industry*, vol. 94, p. 26–40, jan 2018.

[24] M. Brambilla, J. Cabot, and M. Wimmer, “Model-driven software engineering in practice,” *Synthesis Lectures on Software Engineering*, vol. 1, Sep 2012.

[25] Object Management Group Inc., “Omg. mda guide version 1.0.1,” June 2003.

[26] D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, D. Varró, M. Tisi, and J. Cabot, “A research roadmap towards achieving scalability in model driven engineering,” in *Proceedings of the Workshop on Scalability in Model Driven Engineering*, BigMDE ’13, (New York, USA), Association for Computing Machinery, 2013.

[27] S. Mustafiz, O. Hassane, G. Dupont, F. Khendek, and M. Toeroe, "Model-driven process enactment for NFV systems with MAPLE," *CoRR*, vol. abs/1910.11756, 2019.

[28] N. L. S. Fung, S. Kokaly, A. D. Sandro, R. Salay, and M. Chechik, "Mmint-a: A tool for automated change impact assessment on assurance cases," in *Proceedings of the SAFECOMP Workshops*, 2018.

[29] F. Büttner, U. Bartels, L. Hamann, O. Hofrichter, M. Kuhlmann, M. Gogolla, L. Rabe, F. Steimke, Y. Rabenstein, and A. Stosiek, "Model-driven standardization of public authority data interchange," *Science of Computer Programming*, vol. 89, pp. 162–175, 2014.

[30] P. Stevens, "Connecting software build with maintaining consistency between models: towards sound, optimal, and flexible building from megamodels," *Software and Systems Modeling*, vol. 19, 07 2020.

[31] R. B. France, J. M. Bieman, S. P. Mandalaparty, B. H. C. Cheng, and A. Jensen, "Repository for model driven development (remodd)," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 1471–1472, 2012.

[32] F. Basciani, J. Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio, "Mdeforge: An extensible web-based modeling platform," vol. 1242, Sep 2014.

[33] A. Butting, T. Greifenberg, B. Rumpe, and A. Wortmann, *On the Need for Artifact Models in Model-Driven Systems Engineering Projects*, pp. 146–153. Jan 2018.

[34] Ö. Babur, L. Cleophas, M. van den Brand, B. Tekinerdogan, and M. Aksit, "Models, more models, and then a lot more," in *Software Technologies: Applications and Foundations* (M. Seidl and S. Zschaler, eds.), (Cham), pp. 129–135, Springer International Publishing, 2018.

[35] A. Vignaga, F. Jouault, M. C. Bastarrica, and H. Brunelière, "Typing in model management," in *Theory and Practice of Model Transformations* (R. F. Paige, ed.), (Berlin, Heidelberg), pp. 197–212, Springer Berlin Heidelberg, 2009.

[36] J. Bézivin, S. Gérard, P.-A. Muller, and L. Rioux, "Mda components: Challenges and opportunities," Jan 2003.

[37] O. Dayıbaş, Oğuztüzün, and L. Yilmaz, "On the use of model-driven engineering principles for the management of simulation experiments," *Journal of Simulation*, vol. 13, no. 2, pp. 83–95, 2019.

[38] C. Baier and J.-P. Katoen, *Principles of Model Checking*, vol. 26202649. 01 2008.

[39] A. Donzé, "On signal temporal logic," in *Runtime Verification* (A. Legay and S. Bensalem, eds.), (Berlin, Heidelberg), pp. 382–383, Springer Berlin Heidelberg, 2013.

[40] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, *Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*, pp. 135–175. Cham: Springer International Publishing, 2018.

[41] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia, "Logical clustering and learning for time-series data," in *Computer Aided Verification* (R. Majumdar and V. Kunčak, eds.), (Cham), pp. 305–325, Springer International Publishing, 2017.

[42] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd ed., 1999.

[43] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," vol. 16, pp. 423 – 424, 07 2004.

[44] J. Kitchin, "6 - basic statistical inference," in *Statistical Methods for Physical Science* (J. L. Stanford and S. B. Vardeman, eds.), vol. 28 of *Methods in Experimental Physics*, pp. 155–186, Academic Press, 1994.

[45] W. Mertens and J. Recker, "New guidelines for null hypothesis significance testing in hypothetico-deductive is research," *Journal of the Association for Information Systems*, vol. 21, p. 1, 2020.

[46] G. L. Tietjen, *A Topical Dictionary of Statistics*. GBR: Chapman & Hall, Ltd., 1986.

[47] D. Freedman, R. Pisani, and R. Purves, *Statistics: Fourth International Student Edition.* International student edition, W.W. Norton & Company, 2007.

[48] M. F. Triola, *Elementary statistics /.* Boston: Pearson, 9th ed ed., 2004.

[49] N. Bajpai, *Business Statistics.* Pearson, 2009.

[50] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. T. Michaelides, D. R. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. D. Roure, "myexperiment: a repository and social network for the sharing of bioinformatics workflows," *Nucleic Acids Research*, vol. 38, no. Web-Server-Issue, pp. 677–682, 2010.

[51] C. W. Denny Luan, "Experiment." https://experiment.com/, 2017. Accessed: 2017-06-12.

[52] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez, *Modeling in the Large and Modeling in the Small.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[53] S. Efftinge, "Xtend." https://www.eclipse.org/xtend/, 2017. Accessed: 2017-05-12.

[54] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2nd ed., 2009.

[55] TTB Ankara Tabip Odasi, "Verilerle Ankara'nin Sagligi." https://ato.org.tr/files/documents/ATO2019. Accessed on 2021-05-05.

[56] T.C. Saglik Bakanligi, "Covid-19 durum raporu." https://covid19.saglik.gov.tr/TR-68443/covid-19-durum-raporu.html, 2021. Accessed on 2021-05-05.

[57] Republic of Turkey Ministry of Health, "Covid-19 information page, general coronavirus table." https://covid19.saglik.gov.tr/EN-69532/general-coronavirus-table.html, 2021. Accessed on 2021-05-06.

[58] S. Coogan, E. A. Gol, M. Arcak, and C. Belta, "Traffic network control from temporal logic specifications," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 2, pp. 162–172, 2016.

[59] A. Garcia-Dominguez and N. Bencomo, “Non-human modelers: Challenges and roadmap for reusable self-explanation,” in *Software Technologies: Applications and Foundations* (M. Seidl and S. Zschaler, eds.), (Cham), pp. 161–171, Springer International Publishing, 2018.

Appendix A

COMPLETE METAMODEL SPECIFICATION FOR SED-ML

Given SED-ML metamodel is composed of the primary blocks of SED-ML with Hypothesis specification extension in Chapter 4. Note that the metamodel is our interpretation and limited to our requirements for the case studies.

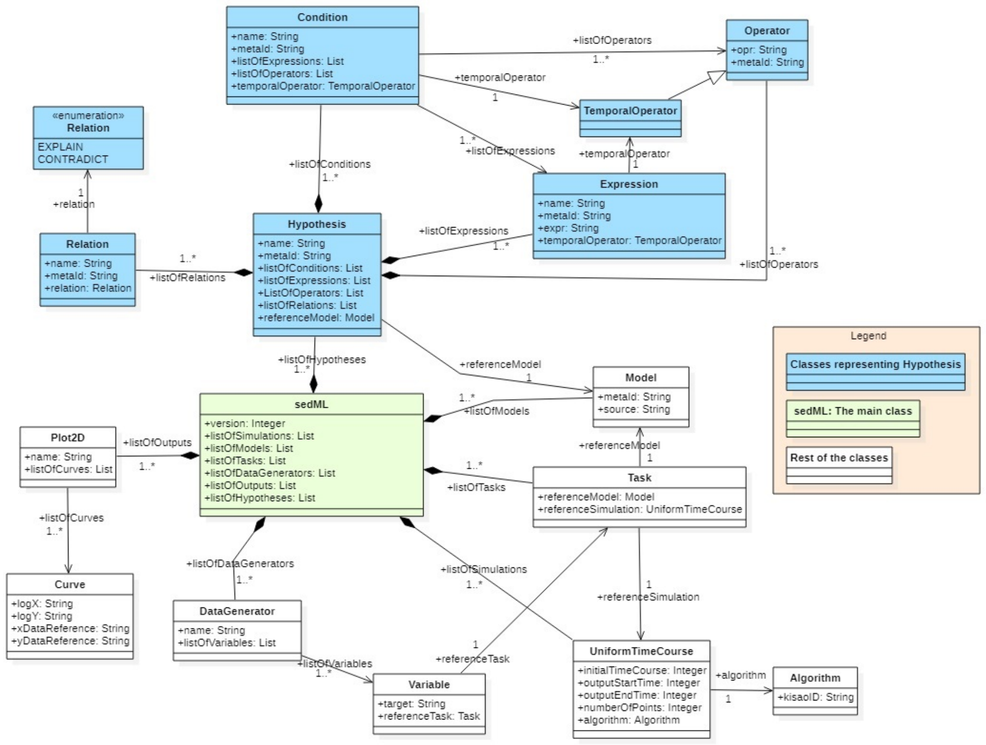


Figure A.1: SED-ML Metamodel

Appendix B

QUADCOPTER QUERYING EXAMPLE

```

class XperimenterQuery {
    def static void main(String[] args) {
            # Get the intervals from the arguments
        double startInterval = args[0]
        double endInterval = args[1]

        # Query the models sequentially
                InputOutput.<String>println("\nXperimenter 1", startInterval,
                ↪  endInterval)
                new XperimenterQuery().queryModel("Quadcopter1.xml")
                InputOutput.<String>println("\nXperimenter 2")
                new XperimenterQuery().queryModel("Quadcopter2.xml")
        }
        # Query the model
        def queryModel(String file, double startInterval, double endInterval) {
                # Register the EMF model
                doEMFSetup
                val resourceSet = new ResourceSetImpl
                # Load the model and get the resources
                val resource = resourceSet.getResource(URI.createFileURI(file),
                ↪  true)
                for (content : resource.contents)
                        validateModel(content, double startInterval, double
                        ↪  endInterval)
        }
        # Validate the model
        def validateModel(EObject o, double startInterval, double endInterval) {
                val expImpl = o as ExperimentImpl
                # find the variable which is between startInterval and endInterval
                for(v: expImpl.design.variables){
                        if (v.lowValue > startInterval && v.highValue< endInterval)
                                InputOutput.<String>println(v.name
                                        +"["+v.lowValue+", "+v.highValue+"] is
                                        ↪  between startInterval and endInterval")
                }
        }
        # Register Xperimenter packages to EMF
        def doEMFSetup() {
                EPackage.Registry.INSTANCE.put(XperimenterPackage.eINSTANCE.nsURI,
                ↪  XperimenterPackage.eINSTANCE);
                Resource.Factory.Registry.INSTANCE.extensionToFactoryMap.put("xml",
                ↪  new XMIResourceFactoryImpl);
        }
}
```

Appendix C

HYPOTHESIS TO SED-ML MODEL GENERATION IN PYTHON

```python
import xml.etree.ElementTree as ET
from XmlOperations import indent

def generateSimulation(listOfSimulations):
    uniformTimeCourseAttr = {"initialTime": "0", "outputStartTime": "0",
    ↪  "outputEndTime": "100", "numberOfPoints": "1000", "id": "simulation"}
    uniformTimeCourses = ET.SubElement(listOfSimulations, "uniformTimeCourse",
    ↪  attrib=uniformTimeCourseAttr)
    algorithmAttr = {"kisaoID": "KISAO:0000088", }
    ET.SubElement(uniformTimeCourses, "algorithm", attrib=algorithmAttr)

def generateHypothesis(listOfHypothesis, optimized_formula1, optimized_formula2,
↪   optimized_formula3):
    hypothesis1 = ET.SubElement(listOfHypothesis, "hypothesis")
    hypothesis2 = ET.SubElement(listOfHypothesis, "hypothesis")
    temporalOp1 = optimized_formula1.split(0, 5)
    exprOp1 = optimized_formula1.split(5, 20)
    conditionAttr1 = {"metaid": "C1", "expr": exprOp1, "temporal operator":
    ↪  temporalOp1}
    temporalOp2 = optimized_formula2.split(0, 5)
    exprOp2 = optimized_formula2.split(5, 20)
    conditionAttr2 = {"metaid": "C2", "expr": exprOp2, "temporal operator":
    ↪  temporalOp2}
    temporalOp3 = optimized_formula3.split(0, 5)
    exprOp3 = optimized_formula3.split(5, 20)
    conditionAttr3 = {"metaid": "C3", "expr": exprOp3, "temporal operator":
    ↪  temporalOp3}

    listOfConditions = ET.SubElement(hypothesis1, "listOfConditions")
    ET.SubElement(listOfConditions, "condition", attrib=conditionAttr1)
    ET.SubElement(listOfConditions, "condition", attrib=conditionAttr2)
    ET.SubElement(listOfConditions, "condition", attrib=conditionAttr3)

    listOfExpressions = ET.SubElement(hypothesis1, "listOfExpressions")
    expressionAttr1 = {"expr": exprOp1, "temporal operator": temporalOp1}
    ET.SubElement(listOfExpressions, "expression", attrib=expressionAttr1)

    listOfRelations = ET.SubElement(listOfHypothesis, "listOfRelations")
    relationAttr1 = {"relation": "CONTRADICT", "hyp": hypothesis1, "hyp":
    ↪  hypothesis2}
    ET.SubElement(listOfRelations, "relation", attrib=relationAttr1)

def generateModel(listOfModels):
    modelAttribute = {"id": "model", "language": "urn:sedml:language:sbml",
    ↪  "source": "urn:miriam:biomodels.db:BIOMD0000000021"}
    model = ET.SubElement(listOfModels, "model", attrib=modelAttribute)


def generateTask(listOfTasks):
    taskAttr = {"simulationReference": "simulation", "modelReference": "model",
    ↪  "id": "task"}
    task = ET.SubElement(listOfTasks, "task", attrib=taskAttr)

def generateDataGenerators(listOfDataGenerators, systemVariables):
    variableTime = 'time'
    datageneratorAttr = {"id": variableTime + 'DG', "name": variableTime + 'DG'}
    datagenerator = ET.SubElement(listOfDataGenerators, "dataGenerator",
    ↪  attrib=datageneratorAttr)

    listOfVariables = ET.SubElement(datagenerator, "listOfVariables")
    timevariablesAttr = {"id": variableTime, "name": variableTime, "taskReference":
    ↪  "task"}
    ET.SubElement(listOfVariables, "variable", attrib=timevariablesAttr)

    mathAttr = {"xmlns": "http://www.w3.org/1998/Math/MathML"}
    math = ET.SubElement(datagenerator, "math:math", attrib=mathAttr)
    ET.SubElement(math, "math:ci").text = variableTime

    for variable in systemVariables:
        variable = 'x' + str(variable)
        datageneratorAttr = {"id": variable + "DG", "name": variable + "DG"}
        datagenerator = ET.SubElement(listOfDataGenerators, "dataGenerator",
        ↪  attrib=datageneratorAttr)

        listOfVariables = ET.SubElement(datagenerator, "listOfVariables")
        variablesAttr = {"id": variable, "name": variable, "taskReference": "task"}
        # ,
        ↪  "target":"/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PX']"
        ET.SubElement(listOfVariables, "variable", attrib=variablesAttr)

        mathAttr = {"xmlns": "http://www.w3.org/1998/Math/MathML"}
        math = ET.SubElement(datagenerator, "math:math", attrib=mathAttr)
        ET.SubElement(math, "math:ci").text = variable

def generateListOfOutputs(listOfOutputs, systemVariables):
    plot2DAttr = {"id": "plot1_Basic"}
    plot2D = ET.SubElement(listOfOutputs, "plot2D", attrib=plot2DAttr)
    listOfCurves = ET.SubElement(plot2D, "listOfCurves")
    for variable in systemVariables:
        variableOut = 'x' + str(variable) + "DG"
        curveAttr = {"id": variableOut, "logX": "false", "logY": "false",
        ↪  "xDataReference": "timeDG", "yDataReference": variableOut}
        ET.SubElement(listOfCurves, "curve", attrib=curveAttr)

def generateSedML(systemVariables, optimized_formula1, optimized_formula2,
↪  optimized_formula3):
    sedmlAttr = {"xmlns:math": "http://www.w3.org/1998/Math/MathML", "xmlns":
    ↪  "http://sed-ml.org/", "level": "1", "version": "1"}
    root = ET.Element("sedML", attrib=sedmlAttr)

    listOfHypothesis = ET.SubElement(root, "listOfHypothesis")
    generateHypothesis(listOfHypothesis, optimized_formula1, optimized_formula2,
    ↪  optimized_formula3)

    listOfSimulations = ET.SubElement(root, "listOfSimulations")
    generateSimulation(listOfSimulations)

    listOfModels = ET.SubElement(root, "listOfModels")
    generateModel(listOfModels)

    listOfTasks = ET.SubElement(root, "listOfTasks")
    generateTask(listOfTasks)

    listOfDataGenerators = ET.SubElement(root, "listOfDataGenerators")
    generateDataGenerators(listOfDataGenerators, systemVariables)

    listOfOutputs = ET.SubElement(root, "listOfOutputs")
    generateListOfOutputs(listOfOutputs, systemVariables)

    tree = ET.ElementTree(root)
    indent(root)

    # writing xml
    tree.write("SEDML_with_hypothesis.xml", encoding="utf-8", l_declaration=True)
    with open("SEDML_with_hypothesis.xml", 'r') as fin:
        print(fin.read())


```

Appendix D

SED-ML TO XPERIMENTER MODEL TRANSFORMATION IN PYTHON

```python
def generateXperimenter(variables, constraints, description=""description""):
    xperimenter = open("xperimenter.xpr", "w")
    xperimenter.write("experiment experiment{")
    xperimenter.write("desc {0};".format(description))
    xperimenter.write("objective COMPARATIVE;")
    xperimenter.write("design design;")
    xperimenter.write("simulation simulation;")
    xperimenter.write("analysis AnovaAnalysis;")
    xperimenter.write("visual DEFAULT;")
    xperimenter.write("target KEPLER;")
    xperimenter.write("\n}")
    varList = ""

    for variable in variables:
        variable = "v"+ str(variable)
        elem = constraints.get(variable)
        if elem is not None and "a"in elem and "b"in elem:
            xperimenter.write("\nvariable {0}: BOOLEAN group
            ↪  RESPONSE;".format(variable))
        else:
            xperimenter.write(
                "\nvariable {0}: INTEGER group FACTOR {1};".format(variable,
                ↪  str(constraints.get(variable))))
        varList = varList + variable + ""

    xperimenter.write("\ndesign design{")
    xperimenter.write("method FULLFACTORIAL;")
    xperimenter.write("varlist %s;"% varList)
    xperimenter.write("\n}")

    xperimenter.write("\nsimulation simulation{")
    xperimenter.write("modelFile data/;")
    xperimenter.write("modelType DISCRETEEVENT;")
    for variable in variables:
        variable = "x"+ str(variable)
        xperimenter.write("inport {0}:  {0};".format(variable))
    xperimenter.write("\n}")

    xperimenter.write("analysis AnovaAnalysis{")
    xperimenter.write("file "http://ceng.metu.edu.tr/~e1564178/xperimenter/anova-service";")
    xperimenter.write("\n}")
    xperimenter.close()

    with open("xperimenter.xpr", "r") as fin:
        print(fin.read())

```

Appendix E

EXPERIMENT EXECUTOR IN PYTHON

```python
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import numpy as np

x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
model = LinearRegression()
model.fit(x, y)
r_sq = model.score(x, y)
print("coefficient of determination:", r_sq)
folder_name = "hospital_bed_data/"

if __name__ == "__main__":
    allData = []
    for file in os.listdir(folder_name):
        filepath = os.path.join(folder_name, file)
        data = []
        with open(filepath, "r") as f:
            if not file.endswith("label") and not file.endswith("properties"):
                data = pd.read_csv(folder_name + "/" + file, sep=" ", header=None)
                data.columns = ["step", "h0", "h1", "h2", "h3", "h4", "h5", "h6",
                ↪  "h7"]
                allData.append(data)

    filename = folder_name + "/system_properties"
    with open(filename) as f:
        content = f.readlines()
    content = [x.strip() for x in content]

    mismatchFormula = content.__getitem__(3).replace("x1", "next")
    if content.__getitem__(3).__contains__(">"):
        formula = content.__getitem__(3).replace(">", "<")

    if content.__getitem__(3).__contains__("<"):
        formula = content.__getitem__(3).replace("<", ">")

    optimized_formula1 = optimized_formula1.replace("=", "==").replace("P 1 1", "")
    optimized_formula2 = optimized_formula2.replace("=", "==").replace("P 1 1", "")
    optimized_formula3 = optimized_formula3.replace("=", "==").replace("P 1 1", "")

    fileIndex = 0
    hypothesisMatchingCount = 0
    totalCount = 0
    mismatchCount = 0
    nullHypothesisCount = 0
    resultMismatchAll= pd.DataFrame(columns=["step", "h0", "h1", "h2", "h3", "h4", "h5", "h6", "h7"])
    for data in allData:
        fileIndex = fileIndex + 1

        data.loc[0, "next"] = 0
        for i in range(1, len(data) - 1):
            data.loc[i, "next"] = data.loc[i + 1, "h1"]

        data.loc[100, "next"] = 0
        totalCount = totalCount + len(data)

                formula = args[0]
        nullHypothesis = data.query(formula)
        nullHypothesisCount = nullHypothesisCount + len(nullHypothesis)

        result = data.query(optimized_formula1 + " | " + optimized_formula2 + " | " + optimized_formula3)

        hypothesisResult = result.query(formula)
        hypothesisMatchingCount = hypothesisMatchingCount + len(hypothesisResult)
        print(hypothesisResult)
        resultMismatch = result.query(mismatchFormula)
        mismatchCount = mismatchCount + len(resultMismatch)

        resultMismatchAll = resultMismatchAll.append(resultMismatch)
        plt.plot(resultMismatchAll["step"], resultMismatchAll["h1"], "d", label="h1")
        plt.plot(resultMismatchAll["step"], resultMismatchAll["h4"], "s", label="h4")
        plt.plot(resultMismatchAll["step"], resultMismatchAll["h6"], "ro", label="j")
        plt.plot(resultMismatchAll["step"], resultMismatchAll["h7"], "o", label="k")
        plt.title("Hypothesis proving values for h1, h4, j and k")
        plt.xlabel("Time")
        plt.ylabel("Value of h")
        plt.xlim(2,400)
        plt.legend()
        plt.show()

    print("HYPOTHESIS PROVING STEP COUNT:", hypothesisMatchingCount)
    print("HYPOTHESIS REFUTING STEP COUNT: ", mismatchCount)
    print("EXCLUDED STEP COUNT: ", totalCount - nullHypothesisCount - mismatchCount)
    print("OVERALL STEP COUNT:", totalCount)


```

Appendix F

EXPERIMENT ANALYSIS TOOL IN PYTHON

```python
import os
import sys
import easygui as eg
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy.stats import stats
from sklearn import linear_model
from statsmodels.formula.api import ols

folder_name = "hospital_bed_data/"

def query(queryString, allData):
    fileIndex = 0
    hypothesisMatchingCount = 0
    totalCount = 0
    for data in allData:
        result = data.query(queryString)
        totalCount = totalCount + len(data)
        hypothesisMatchingCount = hypothesisMatchingCount + len(result)
        plt.plot(result["step"], result["v0"], label="v0")
        plt.plot(result["step"], result["v1"], label="v1")
        plt.plot(result["step"], result["v2"], label="v2")
        plt.plot(result["step"], result["v3"], label="v3")
        plt.plot(result["step"], result["v4"], label="v4")
        plt.plot(result["step"], result["v5"], label="v5")
        plt.plot(result["step"], result["v6"], label="v6")
        plt.plot(result["step"], result["v7"], label="v7")
        plt.title("Dataset " + str(fileIndex) + " for query: " + queryString)
        fileIndex = fileIndex + 1
        plt.legend()
        plt.show()
    print("QUERY MISMATCHING STEP COUNT: ", totalCount - hypothesisMatchingCount)
    print("QUERY MATCHING STEP COUNT: ", hypothesisMatchingCount)
    print("TOTAL STEP COUNT: ", totalCount)


def doHistogram(allData):
    fileIndex = 0
    for data in allData:
        plt.hist(data["v0"].sort_index(), color="blue", edgecolor="black",
        ↪  bins=int(180 / 15), label="v0")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v1"].sort_index(), color="green", edgecolor="black",
        ↪  bins=int(180 / 15), label="v1")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v2"].sort_index(), color="black", edgecolor="black",
        ↪  bins=int(180 / 15), label="v2")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v3"].sort_index(), color="yellow", edgecolor="black",
        ↪  bins=int(180 / 15), label="v3")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v4"].sort_index(), color="orange", edgecolor="black",
        ↪  bins=int(180 / 15), label="v4")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v5"].sort_index(), color="white", edgecolor="black",
        ↪  bins=int(180 / 15), label="v5")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v6"].sort_index(), color="brown", edgecolor="black",
        ↪  bins=int(180 / 15), label="v6")
        plt.xlabel("Value")
        plt.ylabel("Count")

        plt.hist(data["v7"].sort_index(), color="purple", edgecolor="black",
        ↪  bins=int(180 / 15), label="v7")
        plt.xlabel("Value")
        plt.ylabel("Count")
        plt.title("Histogram for dataset " + str(fileIndex))
        plt.legend()
        plt.show()
        fileIndex = fileIndex + 1


def doLinearRegression(allData):
    fileIndex = 0
    for data in allData:
        regr = linear_model.LinearRegression()

        datav0 = np.array(data["v0"].values.tolist())
        datav1 = np.array(data["v1"].values.tolist())

        regr.fit(datav0.reshape(1, -1), datav1.reshape(1, -1))
        plt.scatter(datav0, datav1, color="black")
        plt.plot(datav0, datav1, color="blue", linewidth=3, label="v0 vs v1")

        datav2 = np.array(data["v2"].values.tolist())
        regr.fit(datav1.reshape(1, -1), datav2.reshape(1, -1))
        plt.scatter(datav1, datav2, color="black")
        plt.plot(datav1, datav2, color="orange", linewidth=3, label="v1 vs v2")

        datav3 = np.array(data["v3"].values.tolist())
        regr.fit(datav2.reshape(1, -1), datav3.reshape(1, -1))
        plt.scatter(datav2, datav3, color="black")
        plt.plot(datav2, datav3, color="orange", linewidth=3, label="v2 vs v3")

        datav4 = np.array(data["v4"].values.tolist())
        regr.fit(datav3.reshape(1, -1), datav4.reshape(1, -1))
        plt.scatter(datav3, datav4, color="black")
        plt.plot(datav3, datav4, color="red", linewidth=3, label="v3 vs v4")

        datav5 = np.array(data["v5"].values.tolist())
        regr.fit(datav4.reshape(1, -1), datav5.reshape(1, -1))
        plt.scatter(datav4, datav5, color="black")
        plt.plot(datav4, datav5, color="yellow", linewidth=3, label="v4 vs v5")

        datav6 = np.array(data["v6"].values.tolist())
        regr.fit(datav5.reshape(1, -1), datav6.reshape(1, -1))
        plt.scatter(datav5, datav6, color="black")
        plt.plot(datav5, datav6, color="green", linewidth=3, label="v5 vs v6")

        datav7 = np.array(data["v7"].values.tolist())
        regr.fit(datav6.reshape(1, -1), datav7.reshape(1, -1))
        plt.scatter(datav6, datav7, color="black")
        plt.plot(datav6, datav7, color="brown", linewidth=3, label="v6 vs v7")

        plt.title("Linear regression for dataset " + str(fileIndex))
        plt.legend()
        plt.show()

        fileIndex = fileIndex + 1


def doStudentT(allData):
    fileIndex = 0
    for data in allData:
        resultv0 = stats.ttest_1samp(data["v0"], 0)
        resultv1 = stats.ttest_1samp(data["v1"], 0)
        resultv2 = stats.ttest_1samp(data["v2"], 0)
        resultv3 = stats.ttest_1samp(data["v3"], 0)
        resultv4 = stats.ttest_1samp(data["v4"], 0)
        resultv5 = stats.ttest_1samp(data["v5"], 0)
        resultv6 = stats.ttest_1samp(data["v6"], 0)
        resultv7 = stats.ttest_1samp(data["v7"], 0)
        plt.plot(resultv0, label="v0")
        plt.plot(resultv1, label="v1")
        plt.plot(resultv2, label="v2")
        plt.plot(resultv3, label="v3")
        plt.plot(resultv4, label="v4")
        plt.plot(resultv5, label="v5")
        plt.plot(resultv6, label="v6")
        plt.plot(resultv7, label="v7")
        plt.xlabel("P value")
        plt.ylabel("T statistic")
        plt.title("Histogram for dataset " + str(fileIndex))
        plt.legend()
        plt.show()

        fileIndex = fileIndex + 1


def doStudentZ(allData):
    fileIndex = 0
    for data in allData:
        fileIndex = fileIndex + 1

def doAnovaTest(allData):
    fileIndex = 0
    for data in allData:
        # One - Way ANOVA
        fvalue, pvalue = stats.f_oneway(data["v0"], data["v1"], data["v2"],
        ↪  data["v3"], data["v4"], data["v5"], data["v6"], data["v7"])
        resultOneWay="One way ANOVA: fvalue: " + str(fvalue), " pvalue:
        ↪  "+str(pvalue)
        print (resultOneWay)

        #Two - Way ANOVA
        for variable in data.columns:
            model = ols("{} ~ v0".format(variable), data=data).fit()
            anova_table = sm.stats.anova_lm(model, typ=2)
            print(anova_table)


def doStatisticsSummary(allData):
    fileIndex = 0
    for data in allData:
        resultv0 = str(data["v0"].describe())
        resultv1 = str(data["v1"].describe())
        resultv2 = str(data["v2"].describe())
        resultv3 = str(data["v3"].describe())
        resultv4 = str(data["v4"].describe())
        resultv5 = str(data["v5"].describe())
        resultv6 = str(data["v6"].describe())
        resultv7 = str(data["v7"].describe())
        result = resultv0 + "/n" + resultv1 + resultv2 + "/n" + resultv3 + "/n" +
        ↪  resultv4 + "/n" + resultv5 + "/n" + resultv6 + "/n" + resultv7
        eg.textbox(result, "Summary statistics of dataset " + str(fileIndex))

        fileIndex = fileIndex + 1


if __name__ == "__main__":
    allData = []
    for file in os.listdir(folder_name):
        filepath = os.path.join(folder_name, file)
        data = []
        with open(filepath, "r") as f:
            if not file.endswith("label") and not file.endswith("properties"):
                data = pd.read_csv(folder_name + "/" + file, sep=" ", header=None)
                data.columns = ["step", "v0", "v1", "v2", "v3", "v4", "v5", "v6",
                ↪  "v7"]
                allData.append(data)

    while True:
        try:
            listOfOptions = ["query", "statistical analysis", "exit"]
            question = "Select an analysis option"
            title = "Analysis Tool for Experiment"
            choice = eg.choicebox(question, title, listOfOptions)
            if choice.__contains__("query"):
                queryString = eg.textbox("Enter a query: ")
                query(queryString, allData)
            elif choice == "statistical analysis":
                listOfStatisticalOptions = ["Histogram of datasets", "Statistics summary", "Linear regression", "Student T test", "Student Z test", "Analysis of Variance (ANOVA) test"]
                statQuestion = "Select an statistical analysis option"
                statTitle = "Analysis Tool for Experiment"
                statisticalChoice = eg.multchoicebox(statQuestion, statTitle, listOfStatisticalOptions)
                if statisticalChoice.__contains__("Histogram of datasets"):
                    doHistogram(allData)
                if statisticalChoice.__contains__("Student T test"):
                    doStudentT(allData)
                if statisticalChoice.__contains__("Student Z test"):
                    doStudentZ(allData)
                if statisticalChoice.__contains__("Analysis of Variance (ANOVA) test"):
                    doAnovaTest(allData)
                if statisticalChoice.__contains__("Linear regression"):
                    doLinearRegression(allData)
                if statisticalChoice.__contains__("Statistics summary"):
                    doStatisticsSummary(allData)
            elif statisticalChoice == "exit":
                    sys.exit(0)
        except Exception:
            sys.exit(0)
```