A MODEL DRIVEN ENGINEERING FRAMEWORK FOR SIMULATION EXPERIMENT MANAGEMENT

by

Sritika Chakladar

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 6, 2016

Keywords: Model-Driven Engineering, Domain Specific Language, Design of Experiments, Hypotheses, Model Discovery

Approved by

Levent Yilmaz, Chair, Professor Computer Science and Software Engineering
James Cross, Professor Computer Science and Software Engineering
Saad Biaz, Professor Computer Science and Software Engineering

ABSTRACT

Simulation experiments are a convenient and useful means to gain insight into the operation of scientific models. They are conducted to address specific goals and evaluate specific questions about the model. These simulation models are complex, with many possible factors and outcomes. Also, a model that represents certain key characteristics or behaviors of the system can be analyzed to show the eventual real effects of alternative conditions and courses of action. The strength of simulation is that it enables precisely this “what if” hypotheses analysis, under certain assumptions. Efficient experiment designs are necessary for understanding the impact of these factors and their interactions on the model outcomes that establish the dependencies among goals, hypotheses and experiments with the factors of the model. In our study, we propose a model discovery process by devising questions about the model, designing experiments to validate these hypotheses, executing them, drawing inferences and refining it in an iterative manner to support temporal evidences about the model that have a degree of acceptability of its own. Using the cognitive theory of coherence, we establish links between hypotheses and temporal evidences. We use the principles of model driven engineering and domain specific languages to streamline the discovery process through scientific experimentation.

ACKNOWLEDGEMENT

I sincerely thank my advisor Dr. Levent Yilmaz for all his support, encouragement, patience, and guidance. I would also like to express my gratitude to my advisory committee members Dr. Saad Biaz and Dr. James H. Cross for their participation in my advisory committee and their guidance during my graduate studies. I would like to thank my friend Kyle Doud for his support and help. Finally, I would like to thank my entire family for supporting and believing in me throughout my academic tenure at Auburn.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENT iii

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER 1 1

INTRODUCTION 1

CHAPTER 2 6

BACKGROUND 6

2.1 Domain Specific Language 6

2.2 Experiment Management Systems 8

2.3 Model Driven Engineering 10

2.4 Reproducibility 10

CHAPTER 3 12

SOLUTION 12

3.1 Components of the Goal-Hypothesis-Experiment Framework 12

3.1.1 The Conceptual Level: Goals 13

3.1.2 The Operational Level: Hypotheses 14

3.1.3 The Tactical Level: Experiment 16

3.2 A Computational Strategy to support the GHE Framework 17

3.2.1 Domain 18
3.2.2 Metamodel 19
3.2.3 DSLs for Experiment and Hypothesis Modeling 19
3.2.4 Generative Domain Architecture 21
3.2.5 Reference Implementation 21
CHAPTER 4 22
EVALUATION 22
4.1 Metamodel 23
4.2 BNF 23
4.3 DSL 24
4.4 Reference Implementation 26
CHAPTER 5 29
CASE STUDY 29
5.1 A Domain-Specific Language for the GHE Framework 29
5.1.1 Model 30
5.1.2 Goal 31
5.1.3 Hypotheses 31
5.1.4 Experiment 34
5.1.5 Performance Measure 36
5.2 Code Generation 36
5.3 Application 37
CHAPTER 6 43
CONCLUSION 43

6.1 Summary 43
6.2 Further Topics 44
6.2.1 Coherence/Model Discovery 44
6.2.2 Mechanistic Hypotheses 44
6.2.3 Domain Specific Experiments 45
BIBLIOGRAPHY 46
APPENDIX 51

LIST OF TABLES

Table 1 13

Table 2 15

LIST OF FIGURES

Figure 1 18

Figure 2 23

Figure 3 28

Figure 4 38

Figure 5 38

Figure 6 39

Figure 7 40

Figure 8 40

Figure 9 41

Figure 10 41

Figure 11 42

CHAPTER 1

INTRODUCTION

Computer simulation is a convenient and useful means to gain insight into the operation of scientific models. These models are often very complex, with thousands of factors and many sources of uncertainty. Efficient experiment designs are necessary for understanding the impact of these factors and their interactions on the model outcomes [Sanchez et al. 2014]. Usually, such simulation models are created manually. Close observation helps to get a better understanding of the real-world processes of interest. This is a time-consuming activity, which is likely to be error-prone and lacks credibility, as it is based on human perception of the process. The level of correctness of the simulators, that execute the simulation models, is a significant aspect for evaluating the quality of the simulation. This serves as a motivation to define models at an appropriate abstraction level and accuracy, and to design experiments with substantial information to drive the execution. In order to increase scientific credibility and reproducibility of scientific experiments, it is important to have a complete record of the experimental conditions [Joppa et al. 2013; Merali 2010].

The standards for providing accurate and sufficient record of simulation experiment, keeps evolving [Köhn and Le Novère 2008; Rahmandad and Sterman 2012]. The use of experiment specific languages to effectively address the experiment specification and design, has been widely recognized. Focusing

on the concepts of a particular domain for the development of these modeling languages increases efficiency. Domain specific languages (DSL) are easily read and learned by experts in the field [Consel et al. 2005]. DSLs are useful to domain experts who lack proficiency in programming. They also serve as an efficient tool for reusability [Krueger 1992]. These can be used to record experiment definition and reuse them for reproducibility. In this process, the knowledge integrated in the language is also put to reuse.

Designing and managing experiments in an effective manner is critical to increasing reliability of the simulations [Ewald and Uhrmacher 2014]. However, the under-utilization of the Design of Experiments (DOE) methodology remains a challenge in reducing this credibility gap in simulation studies [Teran-Somohano et al. 2014]. To address these issues, simulation experiment description languages [Ewald and Uhrmacher 2014] and model-driven engineering principles [Teran-Somohano et al. 2015] have proven to be effective in managing simulation experiments.

Scientific experiments are defined with a set of goals and have a purpose to answer certain questions. Its strength lies in analyzing the real effects of alternative conditions and courses of action using “what if” hypotheses (under certain assumptions), on a model representing the fundamental behavior of the system. The use of simulation models for scientific experiments and model discovery has been well established [Teran-somohano et al. 2015; Klösgen 1994; Sliwoski et al. 2014]. In order to produce improved experimentation

practices, we need to establish an appropriate connection among experiments, its objectives and questions related to the simulation model.

In this study, we characterize these dependencies among goals, hypotheses, and experiments within the context of computational discovery. The principles of Model-Driven Engineering (MDE) are used to aid the transformation process and to facilitate the search within the operational level of hypotheses and the tactical level of experiments. Our aim is to demonstrate that the use of MDE strategies coupled with cognitive computing can extend the scope of human intellect and partner with scientists on a broad range of tasks in scientific discovery. These tasks include identifying scenarios, formulating questions, inferring mechanisms, defining or generating experiments designed to answer questions, validating them, drawing conclusions, and evaluating results within an incremental and iterative discovery cycle [Bunge 1998]. This iterative process calls for comprehensive models of hypotheses, experiments, and simulations, along with traceability among them to support the computational discovery process [Sliwoski et al. 2014; Džeroski et al. 2007; Darden 2001].

In our study, we establish a strategy that promotes flexibility in model development while taking into consideration the characteristics of the scientific discovery process using a MDE architecture. This iterative discovery process requires evaluation and revision of numerous assumptions and constraints until sufficient degree of similarity against empirical evidence or targeted behavior is attained. This requires cognitive tools to support the co-evolution of both the

hypothesis and the experiment spaces as active-learning takes place through experimentation.

Our goal is to develop an open-source MDE-enhanced application to design, execute, and analyze simulation experiments. A DSL is introduced towards designing simulation experiments. Additionally, the experiment model is used to test and invalidate the evidences of the simulation model. In this process the user can devise questions to test temporal evidences about the model. Temporal properties that describe the results of the observation have a degree of acceptability of its own. This is followed by the implementation of the experiment specifications in the simulation run.

The advantage of this approach is that it can be used to remove redundancy in a system by identifying and eliminating duplicate models. Experimentation becomes a seamless part of simulation development, by explicit representation of experiment models and hypotheses for the experiments. The approach also facilitates synthesis and execution of experiments along with validation and comparison of the experimentation models. The standardization of the entire process improves reproducibility and reliability of simulation results.

The rest of the thesis is structured as follows. In chapter 2, we present an overview of the existing work on specification of simulation experiments as well as a foundational background for the work. Chapter 3 presents the conceptual framework of the Goal-Hypothesis-Experiment system and sketch the elements of a DSL to illustrate the computational strategy. The experimental results are presented in chapter 4. We present the case study to illustrate the application in

chapter 5. Chapter 6 concludes the thesis and provides an outline for the potential avenues of future research.

CHAPTER 2

BACKGROUND

The Goal-Hypothesis-Experiment framework is developed by employing MDE principles and statistical design of experiments. A simulation model acts as an experimental focus for validating a hypothesis and the replication of the results marks its reliability. But replicability of a simulation model has been an important challenge [Crooks et al. 2008] to support reproducible and replicable scientific knowledge [Teran-Somohano et al. 2014]. In recent years, the efforts toward supporting simulation reproducibility have inspired the use of domain specific languages as the means to express experiment specifications [Schutzel et al. 2015]. Experiment Management Systems standardize simulation experiment specification. MDE concepts centers on the specification of the experiment modeling language as well as its transformation to implementation space.

2.1 Domain Specific Language

According to Van Deursen et al. [2000], the DSL “offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain”. The use of DSL for simulation model description has increased over the years. DSLs allow specification of individual sub-tasks, such as observation, configuration, analysis and evaluation of experimental results. Furthermore, they can be used in a wider scope, to describe the

experiment's goals. The development and research of DSL for experiment specification identifies all of these possible applications.

Even though this idea is relatively new, modeling languages are well established in the field of simulation. Recent work shows that this has been an active area for study and development. However, it is worth mentioning that to a great degree, few efforts in the development of these languages have aimed to cover these goals comprehensively. Some interesting general approaches in standardization of specifications for experiments are discussed below.

The Minimum Information About a Simulation Experiment (MIASE) standard [Köhn and Le Novère 2008] states that in order to promote reproducibility, the executions should contain: (1) the composition of simulation model and its configuration parameters, (2) the conditions for simulation run, (3) the collection method employed during the experiment run, and (4) the result of the run. Rahmandad and Sterman [2012] established distinct simulation experiment recording requirements. The Minimum Model Reporting Requirements (MMRR) standards identify, the default values of the model with their units of measurement and details of the computations in the model, as the minimum requirements specification for simulation experiments. The Preferred Model Reporting Requirements (PMRR) contains record of the data and its source for the model's equations and algorithmic rules, the definition of all model variables, and source code for the model's computational implementation. The Minimum Simulation Reporting Requirements (MSRR) includes recording of the simulation hardware and software platforms, the simulation algorithms used, pre-processing

used to generate input data for the experiment, all the levels applied to factors in the simulation model, the number of iterations of the experiment, and all the post-processing performed on the output data. The Preferred Simulation Reporting Requirements (PSRR) includes information that facilitates the assessment of the results beyond the minimum requirements like random number generation algorithm, confidence levels for estimation etc. Although, these provide powerful reporting standards for experiments, they fail to deal with the issue of complex and abundant data. The use of specification language addresses this challenge effectively.

2.2 Experiment Management Systems

The need for a flexible and powerful Experiment Management System is accelerated by exponential increase in the volume of data, combined with a proliferation of heterogeneous data formats and autonomous systems [Jakobovits et al. 2000]. Experiment management systems specify standards for conducting and managing simulation experiments.

Simulation Experiment Description Markup Language (SED-ML) is an XML-based format which uses MIASE standards for encoding, exchanging and documenting simulation experiments [Waltemath, Adams, Bergmann et al. 2011]. It is used for exchanging experiment descriptions, aiding validation and reuse of simulation experiments. It enables reproducibility of experimentation results with models in the domain of biomedical sciences.

Simulation Experiment Specification via a Scala Layer (SESSL) is a general purpose language defined as an internally defined DSL [Ewald and Uhrmacher

2014]. SESSL allows model specification, definition of replications, the stop condition for simulation run, the objective, and range and optimization method. It is mostly used for specifying and generating rather than describing experiments The SESSL definition is more compact and easier to understand. However, the user should be acquainted with the syntax and semantics of Scala to specify an experiment in SESSL.

Simulation Automation Framework for Experiments (SAFE) [Perrone et al. 2012] standardizes experiment specification to record experiment scenarios and enable reproducibility. Nimrod integrated experiment design tools for efficient execution of the models [Peachey et al. 2008]. The ns-3 Experiment Description Language (NEDL) is an externally defined DSL [Hallagan et al. 2010] and developed to meet the demand for a language capable of explicitly capturing experiment scenarios. The NEDL file specifies a design of experiment space in terms of factors, levels, and constraints that aim to exclude design points that are beyond the interest of the user. It is based on XML and consists of a collection of “elements,” which may be either compulsory or optional in the experiment description. But, it requires special-purpose tools for parsing and document validation which hampers its practical applications. SAFE Language for Experiment Description (SLED) is another externally defined DSL, which overcomes the shortcomings of NEDL. It is based on JavaScript Object Notation (JSON) format which makes it much easier to parse.

2.3 Model Driven Engineering

The MDE principles center on the development of the experiment specification language as well as the transformation rules to map them to implementation space. It addresses the issue of platform dependencies and allows definition of domain concepts effectively. The MDE approach suggests development of a metamodel of the system under study and its transformation into an executable model. Metamodels are abstraction of the model properties. These are used as abstract syntax for the experiment modeling languages. The MDE methodology [Gaševic et al. 2009] provides a framework and strategy to move from the platform-independent experiment domain space to the technical space involving platform-specific executable simulation experiment scripts.

2.4 Reproducibility

Reproducibility refers to the closeness between the results of independent simulations performed with the same methods on identical models but with a different experimental setup [Waltemath, Adams, Beard, et al. 2011]. It is important to keep record of all the experimental conditions in order to reproduce the results of the simulation. This increases the reliability of the simulation experiment. But due to large volume of data and its complexity, it becomes difficult to set reporting standards for reproducibility.

All these approaches fail to explore the relationship between experimental factors, and creating syntax for experiment space search. Existing experiment specification languages do not allow flexibility and language extensibility to address the changing needs of applications. Furthermore, there has not been

sufficient work to explore MDE principles in relation to improving the experiment management system.

CHAPTER 3

SOLUTION

For our study, we place simulation experiments in the context of the scientific discovery process. First, we introduce the Goal-Hypothesis-Experiment (GHE) framework, which helps in structuring the process in terms of conceptual, operational, and tactical levels. This is followed by drawing an outline of a conceptual model-driven engineering architecture to support the framework.

3.1 Components of the Goal-Hypothesis-Experiment Framework

The process starts with the background domain knowledge and involves the following general steps to address a specific goal: (1) Formulate well-structured specific questions. (2) Specify hypotheses from the questions that are developed from the domain ontology to answer the questions. (3) Generate the logical consequences of assumptions in the form of expected behavior. (4) Design computer simulations to test the underlying assumptions (e.g., mechanistic hypotheses) about the phenomena. (5) Validate the simulation for relevance and reliability. (6) Design experiments, execute them, and interpret results. (7) Evaluate the correctness of assumptions, and if necessary revise the model, experiments, or the expected behavior. These steps suggest three major activities, taking place at different levels of abstraction.

3.1.1 The Conceptual Level: Goals

The scientific activity begins with carefully considering the goal of the experiment. The goal specifies the targets to be achieved or phenomenon to be discovered through experimentation. It is specified in relation to a particular context and sheds light on the model under study, the focus of the experiment and the frame of reference or viewpoint. In computational discovery one can aim to characterize, understand, evaluate, predict, or improve the object of the study. The enumeration of the goals in terms of these aspects aids the experimentation and evaluation process.

Aspects	Example
Object of study	Immune system influence on hepatic cytochrome P450 regulation
Purpose	Explain or characterize
Focus	the reason for changes in downstream drug metabolism and hepatotoxicity
Viewpoint	based on the response of hepatic cytochrome P450- regulating mechanisms
Context	when health and/or therapeutic interventions change

Table 1: Goal specification in terms of different aspects

3.1.2 The Operational Level: Hypotheses

After specifying the problem, the solution space is searched to address it. A set of questions are formulated in order to determine the completion of the goal of the study. Hypotheses are generated based on these questions and defined in terms of models of the phenomena or system of interest. The solution consists of assumptions on the model based on earlier observation, experiments or experiences. A hypothesis is a suggested explanation for a phenomenon that can be tested and is based on the experimenter's knowledge and belief of the experiment which are upgraded into laws, resulting in a system of laws, called theories.

With respect to model-driven generation and simulation-based knowledge, we identified the following types of hypotheses:

- *Phenomenological hypotheses* generally represent a resultant behavior or output of the system triggered by the change in input conditions of the model. It addresses the impact of input factors (independent variables or control variables) on the output (dependent variables) of the model, under a set of constraints. Such hypotheses allow comparing system configurations, performing sensitivity analysis, and conducting Analysis of Variance (ANOVA) to study system performance.
- *Mechanistic hypotheses* define the mechanisms that generate specific behaviors in the model. Experiments are designed to provide evidence to either support or refute the explanation of the behavioral mechanism defined in the hypothesis.

Type	**Hypothesis**
Phenomenological	In response to lipopolysaccharide, Kupffer cells down regulate hepatic P450 levels via inflamatory cytokines, thus leading to a reduction in metabolic capacity.
Mechanistic	Inflammatory induced P450 down-regulation is mediated by proinflammatory cytokines that specifically regulate different yet overlapping subsets of P450s in both humans and rats [Aitken and Morgan 2007]. Many of these cytokines are derived from Kupffer cells. While some cytokines down-regulate P450 in primary hepatocytes cultures, others are dependent upon the presence of Kupffer cells [Sunman et al. 2004]. Kupffer cells can be activated by bacterial endotoxin (lipopolysachharide, LPS). An LPS stimulus causes Kupffer cells to release proinflammatory cytokines, triggering

	P450 down-regulation and the subsequent decrease in drug clearance.

Table 2: Types of hypothesis

3.1.3 The Tactical Level: Experiment

This level drives the design, execution, and adaptation of an experiment to answer the questions and verify, validate, or refute the assumptions and hypotheses. Outcomes of experiments feedback into the process to facilitate revision of goals, models, questions, and experiments. An experiment may have a set of responses, factors, and a range of its values called factor levels. These are specified by the experimenter and are updated, if needed, to enable adaptation.

The GHE framework serves as a tool for specifying and interpreting operational questions and tactical experiments for conceptual research goals. It allows the definition of the goal of study. A set of hypotheses (formulated in the form of questions and assumptions) are devised to address the goal of the experiment, which refine the issue underlying the problem into its major components. These questions are translated to experiment designs to validate or invalidate them. Experiments are designed and executed in a way to discriminate between rival hypotheses. Within the current state of the art, simulation tools and techniques are not structured to support seamless navigation and traceability between these levels. To mitigate this issue, we propose a computational strategy that supports

the GHE framework by leveraging principles and practices of model-driven engineering and domain-specific languages.

3.2 A Computational Strategy to support the GHE Framework

MDE has emerged as a practical and unified methodology to alleviate the complexity of platforms and express domain concepts effectively [Schmidt 2006]. The use of platform independent domain models along with explicit transformation models facilitates deployment of simulations across a variety of platforms. While the utility of MDE principles in simulation development is now well recognized, its benefits for experimentation have not yet received sufficient attention.

A conceptual framework that integrates MDE, agent models, and product-line engineering to manage the overall lifecycle of a simulation experiment is presented in Figure 1. In the component architecture, the experiment and simulation model spaces are tightly coupled to orchestrate the co-evolution of simulation and experiment spaces as learning takes place. Next, we review these components to open a discussion about their potential contributions to the process of computational discovery.

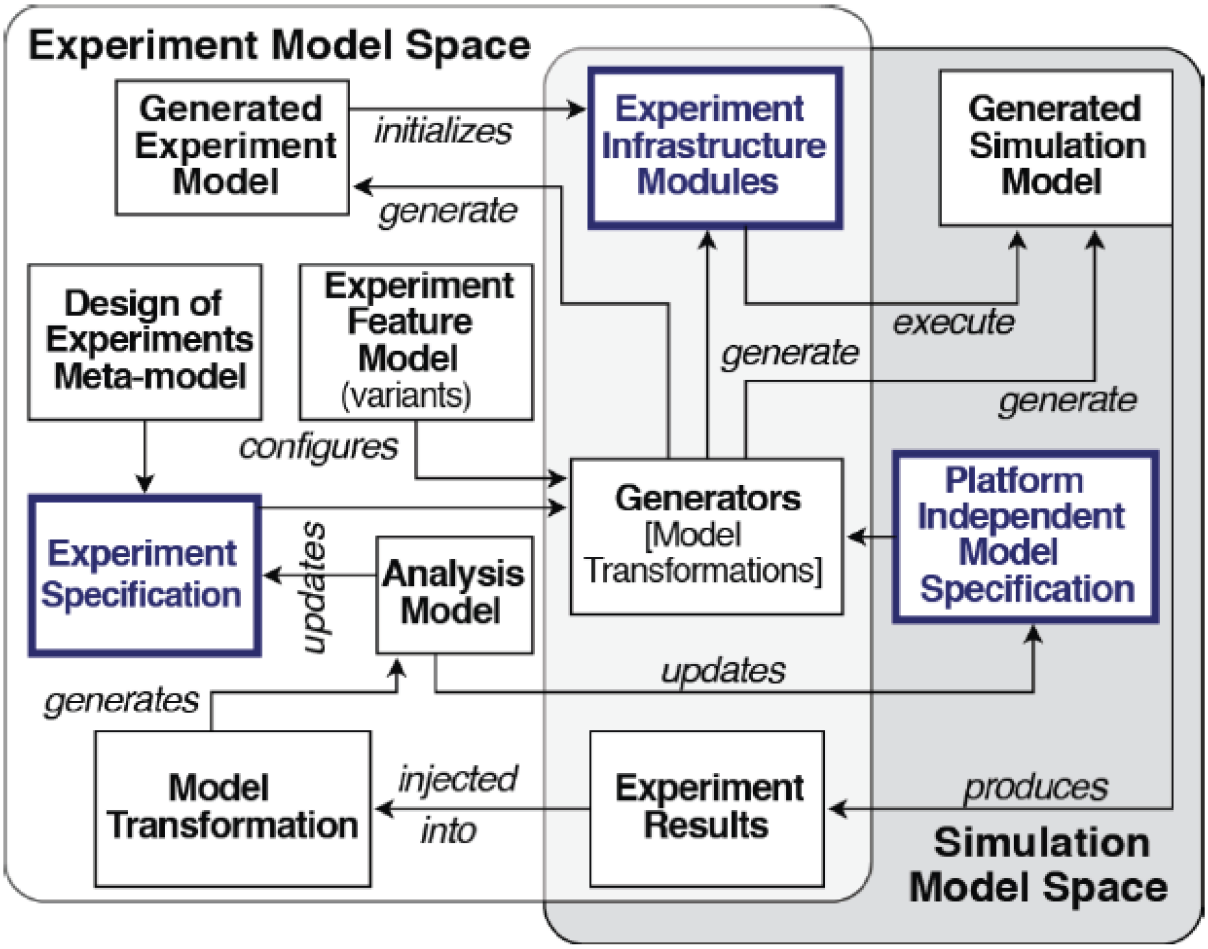


Figure 1: Experiment Management Framework

3.2.1 Domain

Domain refers to a bounded field of interest or knowledge. [Völter et al. 2013] The domain for our study is experiment management. It is useful to develop an ontology by identifying all the relevant concepts of the domain of interest. The ontology represents knowledge about the elements that form an experiment. It encompasses the structural elements of an experiment, including the experiment's goals through all the iterations and questions about the model. It also consists of the inputs to the experiment model and the desired outputs.

3.2.2 Metamodel

Metamodel is an abstract representation of a system's structure, function or behavior. In the context of Model Driven Software Development (MDSD), it is necessary to be clear about the structure of a domain (i.e., its ontology), so that formalization of this structure or its relevant part is possible. [Völter et al. 2013] The metamodel is a basic UML representation of all the relevant concepts of the domain. It comprises of the abstract syntax and the static semantics of a language.

3.2.3 DSLs for Experiment and Hypothesis Modeling

For generating experiment specifications from research questions and hypotheses, the DOE methodology in simulation experiment design [Kleijnen et al. 2005; Sanchez et al. 2014]could provide a structured basis for automation. The ontology defines the vocabulary and grammar. i.e., the abstract syntax for building the experiment domain model. To support the instantiation of the experiment specifications conforming to the DOE metamodel, a suitable DSL is needed.

DSLs are widely used in simulation studies as tools to describe the model. In recent years, the efforts toward supporting simulation reproducibility have inspired the use of DSLs as the means to express model specifications [Darden 2001]. Even though this idea is relatively new, modeling languages are well established in the field of simulation. Recent work shows that this has been an active area for study and development.

The research and development of DSL for the GHE framework, identifies many possible applications. DSLs help to state mechanisms and define parameters along with their properties, in the model. It also helps in identifying sub-tasks in the experimentation procedure, such as observation, configuration, analysis and evaluation of experimental results. Furthermore, they can be used in a wider scope, to describe formally the hypotheses about the model and list evidences derived from the real life experiments.

The experiment model defined by the DSL needs to be configured with the aspects specified in an experiment feature model. An experiment design can have various mandatory and optional features. Features are prominent attributes that facilitate modeling variants of experiments to support different objectives. For instance, the type of the experiment design (e.g., factorial, fractional factorial) and the analysis method (e.g., ANOVA vs. MANOVA) are potential features that collectively define plausible configurations of an experiment.

Advantages of using a DSL:

- To increase accessibility to perform complex computation in the background.
- To increase conciseness and expressiveness in the experiment specification and design generation.
- To increase flexibility and language extensibility in order to accommodate the required changes in the application.

3.2.4 Generative Domain Architecture

An experiment design agent evaluates questions of interest to generate an experiment design that is effective in discriminating rival hypotheses and efficient in covering the parameter space of the system. A trade-off analysis between the number of design points and the number of replicates per design point are carried out in relation to the type of experiment being conducted.

This generative architecture is also used to derive templates for generating the transition of mechanisms from the hypothesis space to the implementation space. A text-model transformation takes place that generate code templates to replicate the phenomenon represented by the hypotheses.

3.2.5 Reference Implementation

Reference implementation represents the concrete realization of the architectural aspects. It contains all implementation details of the semantics of the architecture-centric UML profile constructs on the source code level [Völter et al. 2013]. The templates for the generative architecture are derived from this implementation. In our study, the reference implementation consists of the realization of the concepts of the experiment ontology as well as use-cases that demonstrated the application and the transition from model to implementation space.

CHAPTER 4

EVALUATION

The principles of model-driven software development are used throughout the development process. We started by developing a metamodel for the language in the form of a UML class model in order to facilitate understanding of the domain. We used this metamodel as a roadmap to develop a context-free grammar in Back-Naur Form (BNF). Next, we transformed this BNF grammar into an Xtext grammar and evaluated its readability in a reference model. The grammatical constructs defined in the Xtext grammar were used to identify classes and structures for a reference implementation, where a use-case for the application was developed and tested. Through development of the reference implementation, we were able to identify sections of code that were candidates for text-to-model transformation. These transformations bridge the gap between reference model, reference implementation, and platform.

The process was an effective tool for streamlining the development of a DSL-driven application. By focusing on the way the language will be used before the implementation, we were able to create a highly expressive language while providing support for platform versatility.

4.1 Metamodel

The metamodel encompasses all the major components of the GHE framework. It includes the goal of the experiment, model definition, hypothesis and an experiment. The metamodel for our study is shown below.

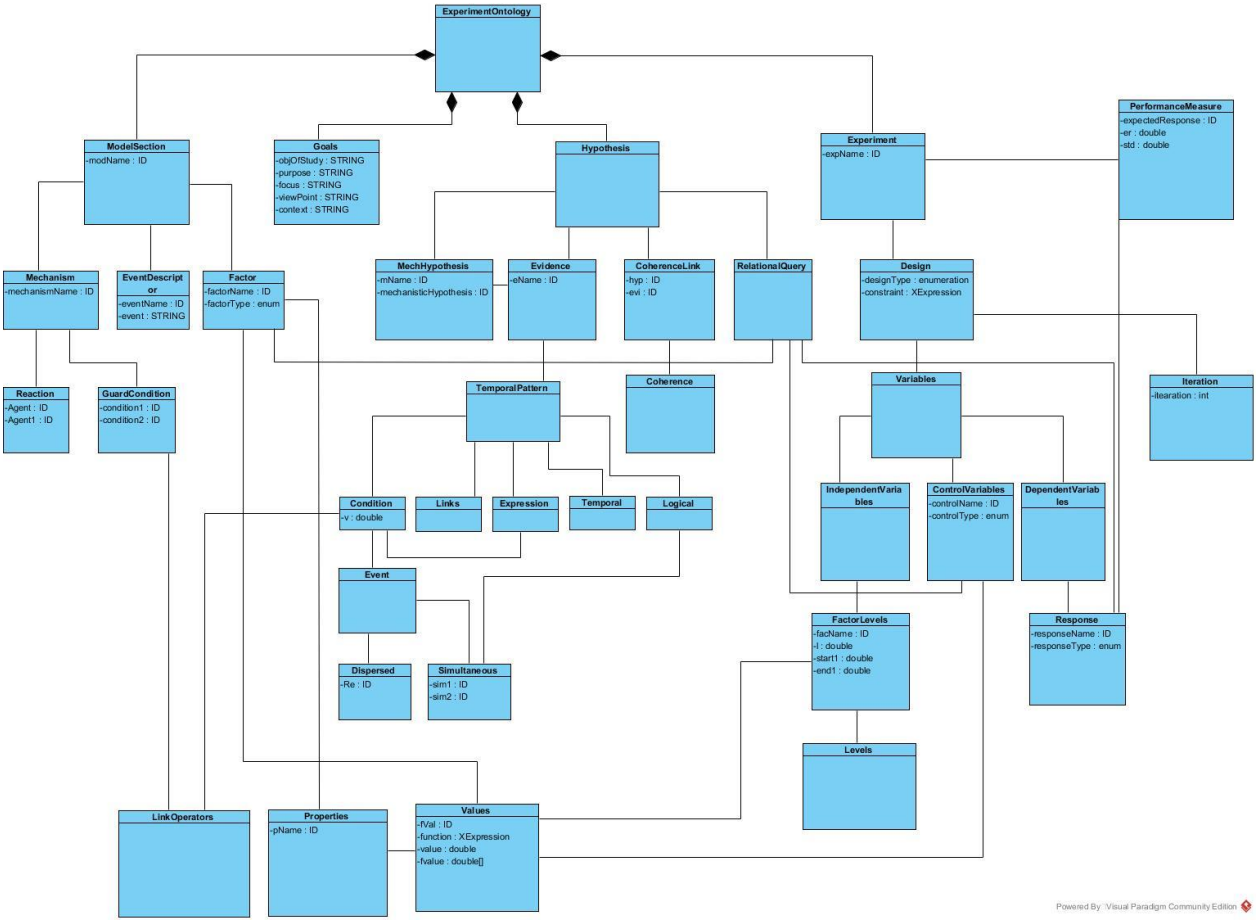


Figure 2: Metamodel representing major components of the GHE framework

4.2 BNF

The first step to define this new language was to describe the syntax in an easily readable/writable format. The BNF notation is used to define the syntactic grammar. In standard BNF, a grammar is defined by a set of terminal and non-terminal symbols. We defined the grammar in the standard form, without use of

extended BNF symbols like *, +, -, etc., because those aspects, while making development easier, make the grammar harder to read. Since the purpose of the BNF development was to discover how the language should look, the most easily interpreted form seemed like the best choice.

The BNF grammar definition during the early stages of development is listed below:

<Model> ::= ExperimentOntology

<ExperimentOntology> ::= ModelSection | Goals | Hypothesis | Experiment

<ModelSection> ::= model <id> {<Mechanism> <EventDescriptor> <Factor>}

<Mechanism> ::= mechanism <id> = <Reaction> <GuardCondition> -> <Reaction>

The full definition of the BNF grammar can be found in the Appendix.

The next step in development was to implement the BNF grammar using a language engineering framework. We proceed by implementing the grammar in Xtext.

4.3 DSL

The DSL for simulation experiment model development is developed using the Xtext DSL development environment on Eclipse Kepler, by translating the experiment ontology metamodel. The DSL is also used to define a set of

hypotheses for experiment model validation and verification. The simulation experiment specifications are then used for the description of a simulation experiment. After the generation of the experiment model design with all the elements imposed by the experiment ontology, it is transformed and stored in a properties file. This file is then used to run the MASON model and collect the results of the simulation run.

The strategy is to develop a grammar to help the user specify experimentation parameters and to verify the conditions to trigger them and determine a desired plan of action. The result of the action is then translated to the user after performing a set of validation and verification.

The transition from BNF to an Xtext implementation is straightforward. Each non-terminal in BNF is treated as a grammar rule, and terminals are either IDs or new keywords. As a simple example, the following grammar rules in Xtext correspond to the BNF's transition shown in the previous section:

```
ExperimentOntology :
        ModelSection |Goals | Hypothesis | Experiment
;
ModelSection:
        'model' (modName = ID)
        '{'
        (mechanisms += Mechanism)*
        ((events += EventDescriptor)?)*
        (parameters += Factor)*
```

'}'

;

Mechanism:

'mechanism' (mechanismName = ID) ' = ' (LHS = Reaction)

(condition = GuardCondition)? ' -> ' (RHS = Reaction)

;

The full listing of the Xtext grammar definition can be found in the Appendix.

4.4 Reference Implementation

The reference implementation for this model is a java program that serves as an implementation of the rules from the grammar as java classes. As we developed the program, we discovered that some aspects of the metamodel that were used in the grammar were simply textual devices for readability and served no purpose for computation. These aspects were subsumed as identifiers in text recognition algorithms for the relevant classes. Some of the use-cases are described below.

- Usecase1.java

This class initializes the pieces of an experiment specification and executes the experiment with the specified parameters.

- ToDeliveryProperties.java

In this class we construct the delivery.properties file from the factors and their values specified by the user. This file is later used for the MASON simulation run.

- ToISHCProperties.java

In this class we construct the ishc.properties file from the factors and their values specified by the user. This file is later used for the MASON simulation run.

- Query.java

In the Query class, a string that represents a temporal property is passed to the constructor and is set as a global variable. The constructor calls a method, detectEvents(), which in turn calls detectPattern() and detectPostfix(). The objective of these functions is to discover which Linear Temporal Logic (LTL) formula matches the sentence from the grammar and to find the conditions that will be inserted into the formula. This class subsumed most of the temporal specification keywords as well as Conditions.

- ConvertToLTL.java

The purpose of this class is to take the events identified from the Query class and replace them in a matching LTL formula that can be found in a patterns.xml file. An LTL formula in the XML file would be in a form like: [](Q & !R -> (!P W R)), where each of the letters (aside from W, which represents “weak until” in temporal logic), represents a placeholder for a condition from the DSL. The difficulty in textually substituting these letters for their condition identifiers was due to the fact that the conditions could have the same capital letters in them as the placeholders in the formula, causing unexpected results.

- ExperimentExecuter.java

This class runs the model and gives the output.

The class diagram generated from the reference implementation, shown in figure 2, resembles the metamodel generated in the first step of our development process.

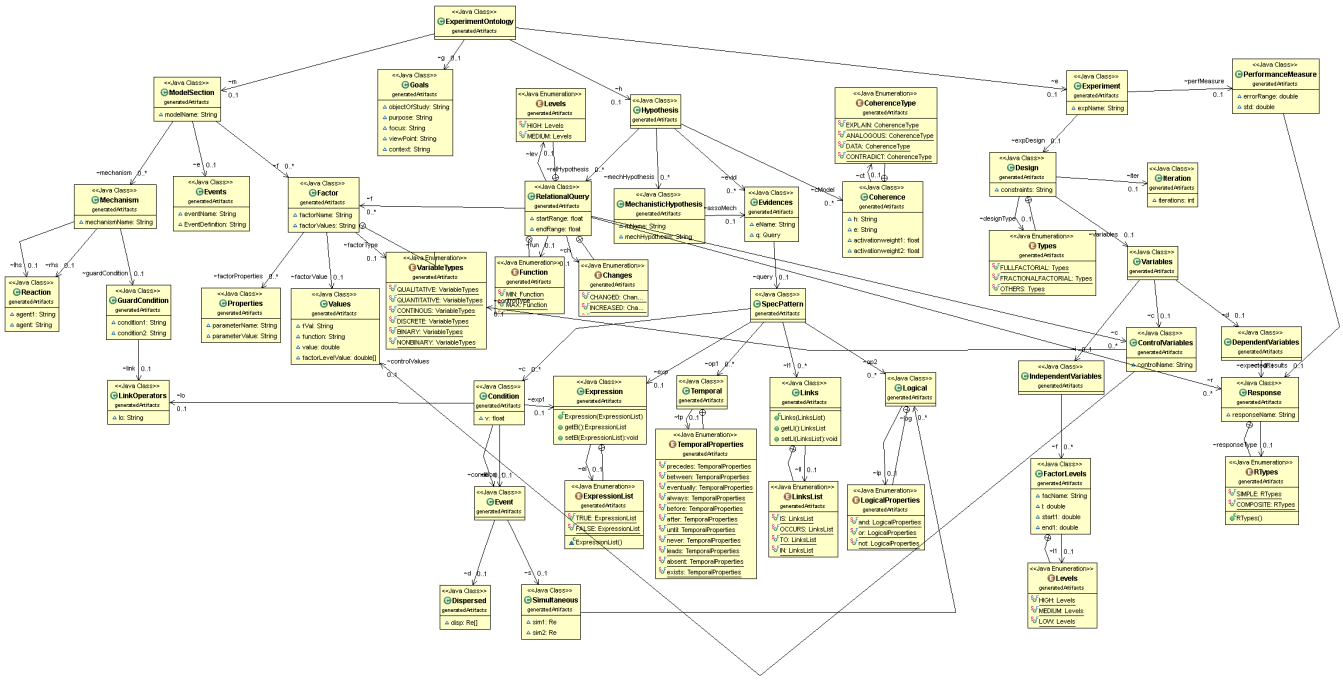


Figure 3: Class diagram generated from the reference implementation

CHAPTER 5

CASE STUDY

5.1 A Domain-Specific Language for the GHE Framework

Our Model-Driven approach for experiment management is driven by explicit specification of goals, hypotheses, and experiments. We studied and modeled different types of hypotheses which allow the user to ask questions about the model or the system under study. In the context of DOE, hypotheses can be defined as mechanistic hypotheses, relational hypotheses, and constraints. For illustration purposes, the evolving DSL is used to define experiments for an agent-based In Silico Hepatocyte Culture (ISHC) model [Petersen et al. 2014], which we replicated to illustrate the proposed concepts in this study.

In order to test the validity of our framework and the practical utility of the approach, we used our project to demonstrate the ISHC model. The DSL we developed is abstract and free of any technical terms. The DSL covers all relevant concepts of the domain with language elements. All schematically-implementable code fragments of the reference implementation are covered by constructs of the DSL. The reference ISHC model is an instance of the DSL. The DSL for simulation experiment model is developed by mapping the experiment ontology metamodel.

5.1.1 Model

Model consists of a specification about the model's name, the mechanisms, the events and the factor parameters. Mechanisms consist of the processes which is assumed to take place in the simulation system. Events define the path for tracing the functions that evaluate the events that form a part of the evidences. Parameters are the inputs to the model and their properties, which have an impact in determining the response/output of the simulation run.

model ISHC{

mechanism M1 **=** inflammatoryAgent **+** Kupffercells [inflammatoryAgent > inflammatorythreshold] **->** Cytokines

mechanism M2 **=** inflammatoryAgent **+** Kupffercells [noOfCytokine > cytokineThreshold] **->** Cytokines

event inflammation **=** 'void ishc.model.KupfferCell.handleInflammation()'

parameter LPS **=** Solute **with properties** {tag: LPS, bindable: true, bolusRatio:1.0 , pExitMedia: 0.1 , pExitCell: 1.0 , bindProb : 0.25 , bindCycles : 1 , numProps : 8 , membraneCrossing: true, bileRatio : 0.5 , core2Rim : 0.50 , metProbStart : 0.3 , metProbFinish : 0.3 , metabolites: 'LPS-Metabolite_A', inflammatory : true , pDegrade : 0.0}

parameter forwardBias **= DISCRETE with values** {0.5}

}

5.1.2 Goal

Goals define what the purpose of the experiment is. It also gives an idea about the specific field of concern and the context under which the study is performed.

goal

{

object of study : 'Immune system influence on hepatic cytochrome P450 regulation'

purpose : 'Explain / characterize'

focus : 'the reason for changes in downstream drug metabolism and hepatotoxicity'

view point : 'based on the response of hepatic cytochrome P450-regulating mechanisms'

context : 'when health and/or therapeutic interventions change.'

}

5.1.3 Hypotheses

Hypotheses consists of relational hypotheses, mechanistic hypotheses and expected regularities. Mechanistic hypotheses deal with the effect of changes in the mechanism of the model. Relational hypotheses deal with the impact of changes in inputs or outputs. In order to represent behavioral changes in the model, we focus on mechanistic hypotheses for the study. Expected regularities are the temporal properties that are to be verified in the experimental run. It is stated in terms of factors and their properties.

The coherence model describes the explanatory coherence relation [Thagard 1989] between the hypothesis and the evidence. The evidence can have an activation weight which indicates its reliability. This is used to establish the weightage of the link between the evidence and hypothesis in the coherence network. We identified the explanatory coherence concept that would be relevant in our framework and would help in discovering the model mechanisms. To summarize how we will be implementing explanatory coherence theory we listed the key terms from the principles, which was used in the experiment definition.

- EXPLAIN

We use this if a coherence exists which explains or supports evidence(s) and hypothesis(es). In this case excitatory links are established between the evidences and hypotheses and an activation weight is assigned to each link. As the number of such links between the hypotheses and evidences increase, the weight on the links in the network decreases.

- ANALOGOUS

We use this if hypothesis and evidence are analogous to each other. Analogy, produces excitatory links between the similar evidences and hypotheses and an activation weight is assigned to each link.

- DATA PRIORITY

The principle of data priority is used to set up explanation-independent excitatory links to each data unit from a special evidence unit that always has an activation of 1. The data units can have activation level specified depending upon its

reliability index. When the network runs, activation spreads from the special evidence unit to data units and then to the units representing explanatory hypotheses.

- CONTRADICT

We use this if there is incoherence between the evidence(s) and hypothesis(es). In this case inhibitory links are established between the evidences and hypotheses and a negative weight is assigned to each link. As the number of such links between the hypotheses and evidences increase, the weight on the links in the network changes.

The conditions are grouped under these categories which are used in designing a query based DSL to allow the user to define the hypothesis which can be used to develop a simulation model.

hypotheses

{

mechanistic hypotheses

{

H1 : M1 **occurs before** M2

}

evidence

{

E1: inflammation **occurs after** inflammatoryAgent >

inflammatoryAgentThreshold

activation weight : 0.5

E2: inflammation **is absent after** cytokine <

cytokineThreshold

activation weight : 0.5

}

coherence model

{

EXPLAIN (H1)(E1)

DATA (Experiment1)(E1 E2)

}

}

5.1.4 Experiment

The ontology for the experiment section encompasses the structural elements of an experiment which includes the experiment's design and performance measure. Based on the model's parameters and their levels, the hypotheses and goal of the experiment, a design is created that is used in subsequent steps of the experiment life-cycle.

The experimental design is defined by the dependent variables, the control variables, the independent variables and their levels, constraints and values which in turn are mappings of the variables provided by the user. Based on this design, one can define what is known as a design matrix, which

specifies the actual experimental runs, that is, the combination of factor levels.

```
experiment Experiment1
{
    design
    {
        designType FULLFACTORIAL
        variables
        {
            independent variables
            {
                LPS are at levels : LOW where LOW is in the
                range 1.0 to 1.0
                TOL are at levels : LOW where LOW is in the
                range 1.0 to 1.0
                DZ are at levels : LOW where LOW is in the
                range 1.0 to 1.0
            }
            dependent variables
            {
                cytokines : type SIMPLE
            }
        }
```

```
        }
    }
```

5.1.5 Performance Measure

An experiment consists of performance measure parameters which defines the criteria for successful experimental run. Basing on this measure we can decide whether additional iterations are required for satisfying the experiment's objective. It is defined in terms of the expected value of the response or output of the experiment and its standard deviation.

```
    performance measure is
        {
            cytokines= 500 +-10
        }
```

In the above example, the expected value of the cytokines after successful experiment execution is 500 with a standard deviation of 10.

5.2 Code Generation

We used the Xtend code generation process for mapping the DSL to platform. A set of templates were derived from the reference implementation and used for the transformation step.

```
    class DOEGenerator implements IGenerator {
        override void doGenerate(Resource resource, IFileSystemAccess
    fsa) {
```

```
fsa.generateFile('ishc.properties',
toISHCProperties(resource.allContents
        .filter(typeof(ModelSection)).head))
fsa.generateFile('delivery.properties',
toDeliveryProperties(resource.allContents
        .filter(typeof(ModelSection)).head ,
    resource.allContents.filter(typeof(Experiment)).head))
fsa.generateFile("KupfferCell.java",
toKupfferCell(resource.allContents
        .filter(typeof(ModelSection)).head))
fsa.generateFile("Hepatocyte.java",
toHepatocyte(resource.allContents
        .filter(typeof(ModelSection)).head)) }
```

5.3 Application

We developed an application to demonstrate our framework and its functionalities. The experiment specification defined using the DSL and the generated artifacts were used to run the ISHC simulation model in MASON to get the results. The figures below illustrate various functions supported by the application.

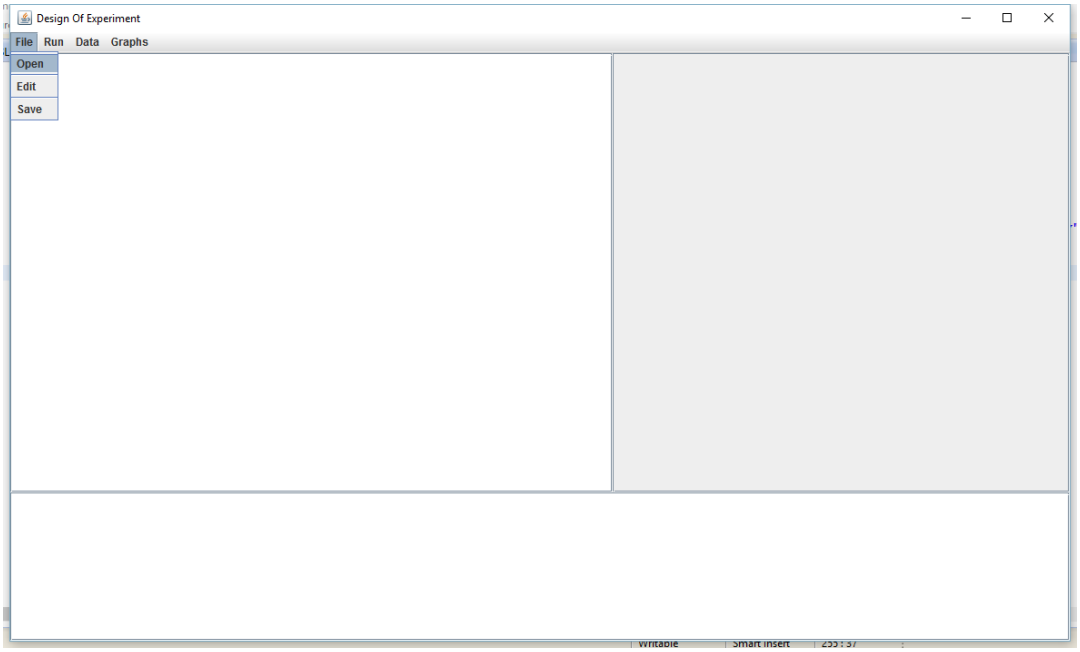


Figure 4: The File menu and its submenus

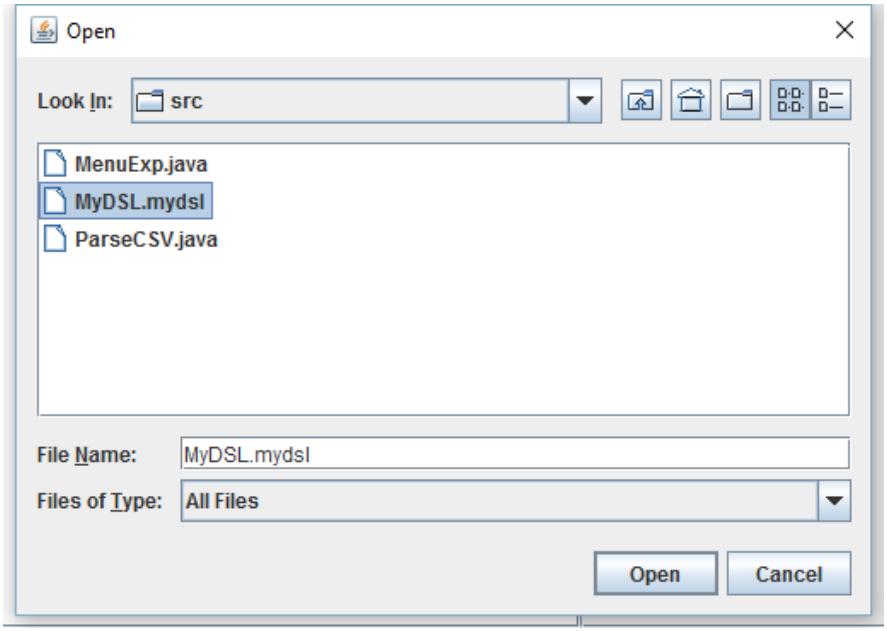


Figure 5: The file explorer opens on clicking the Open menu

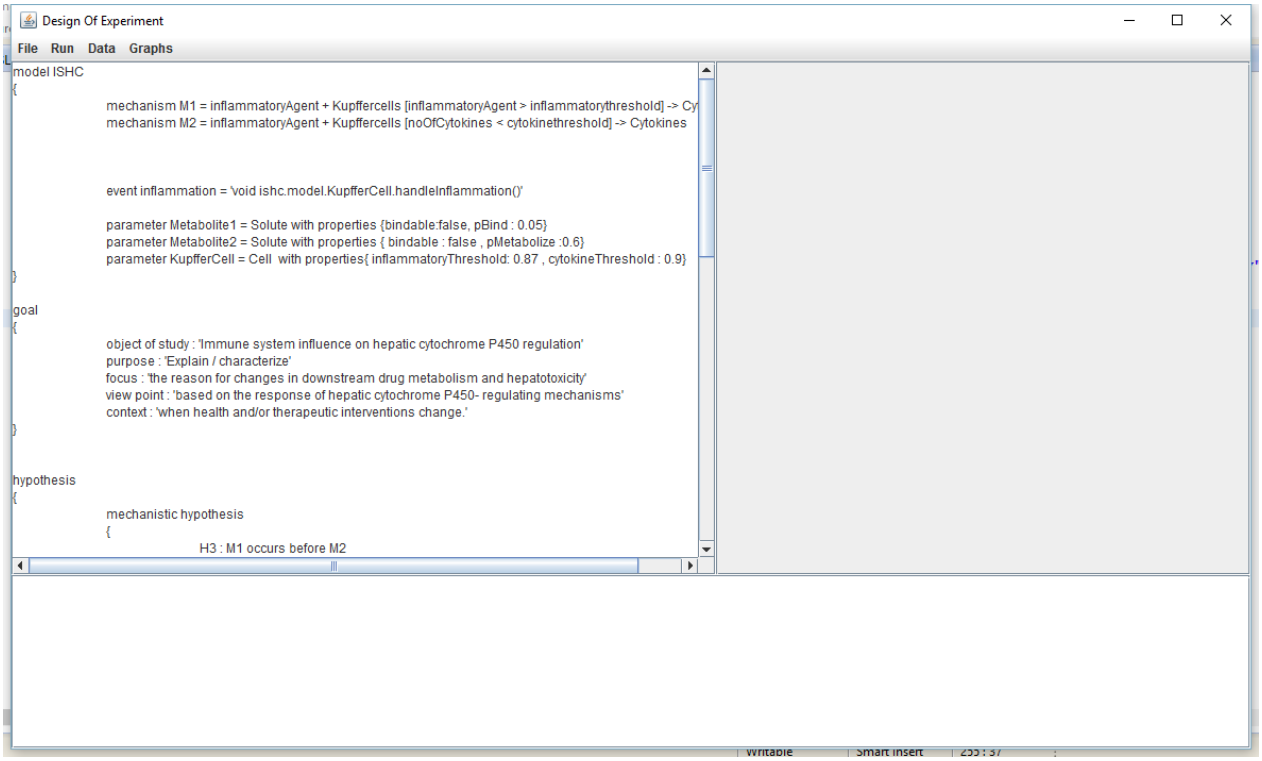


Figure 6: Upon selecting the Xtext file from the specified location, the file opens up and appears in the text area

We can also edit the file with the required changes and save it in a desired location.

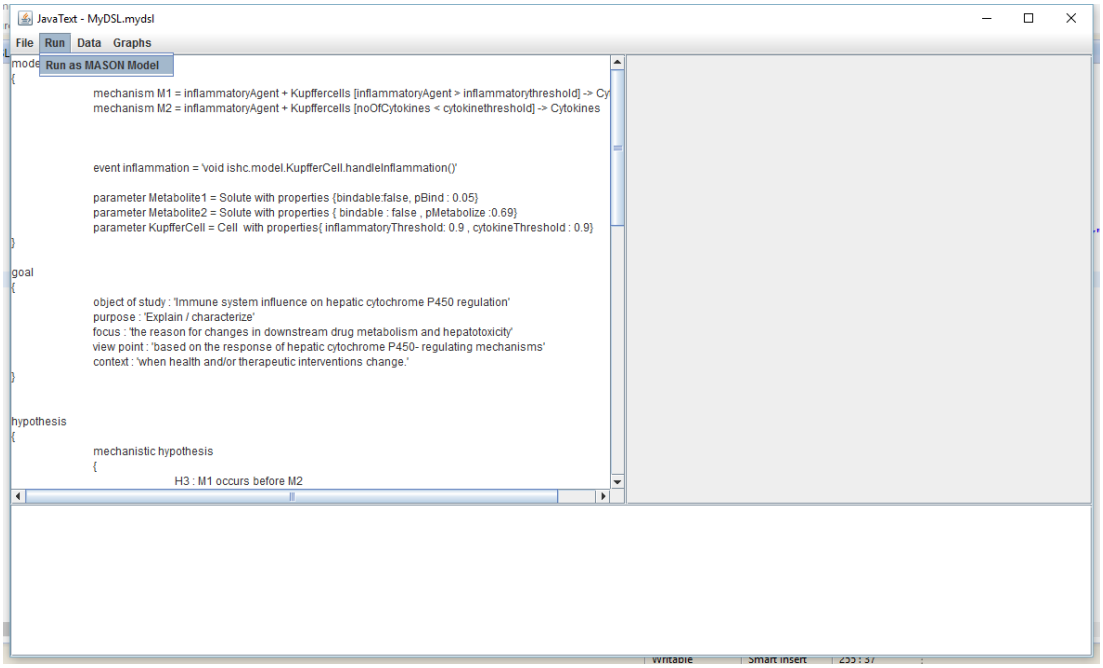


Figure 7: Run the MASON model using the specification of the Xtext file

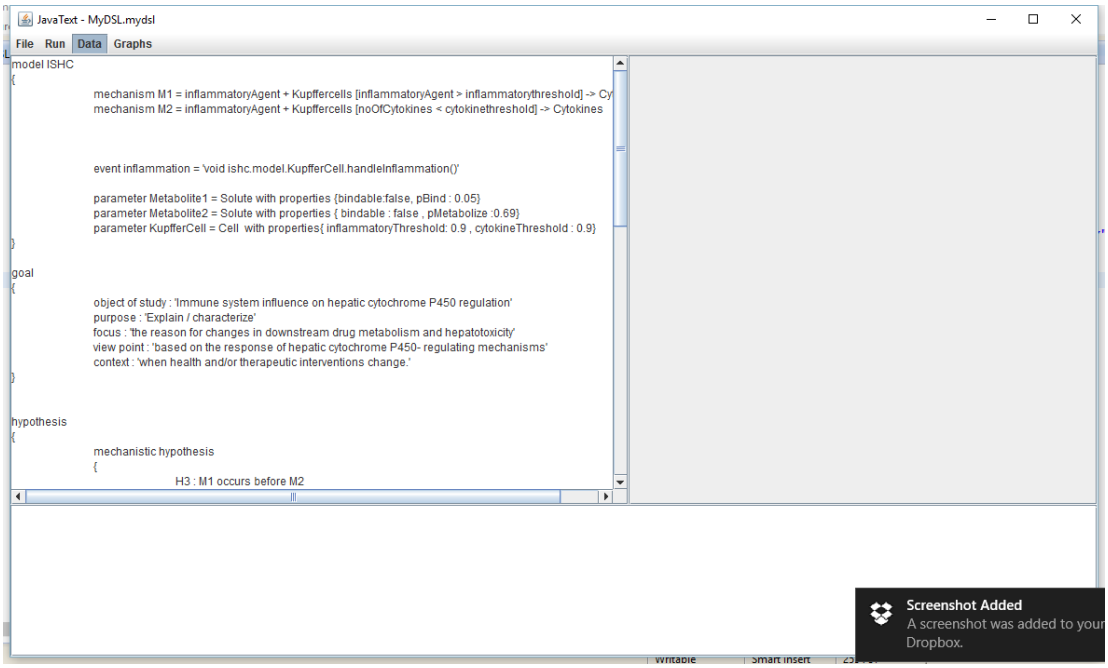


Figure 8: Data Menu displays the result of the run

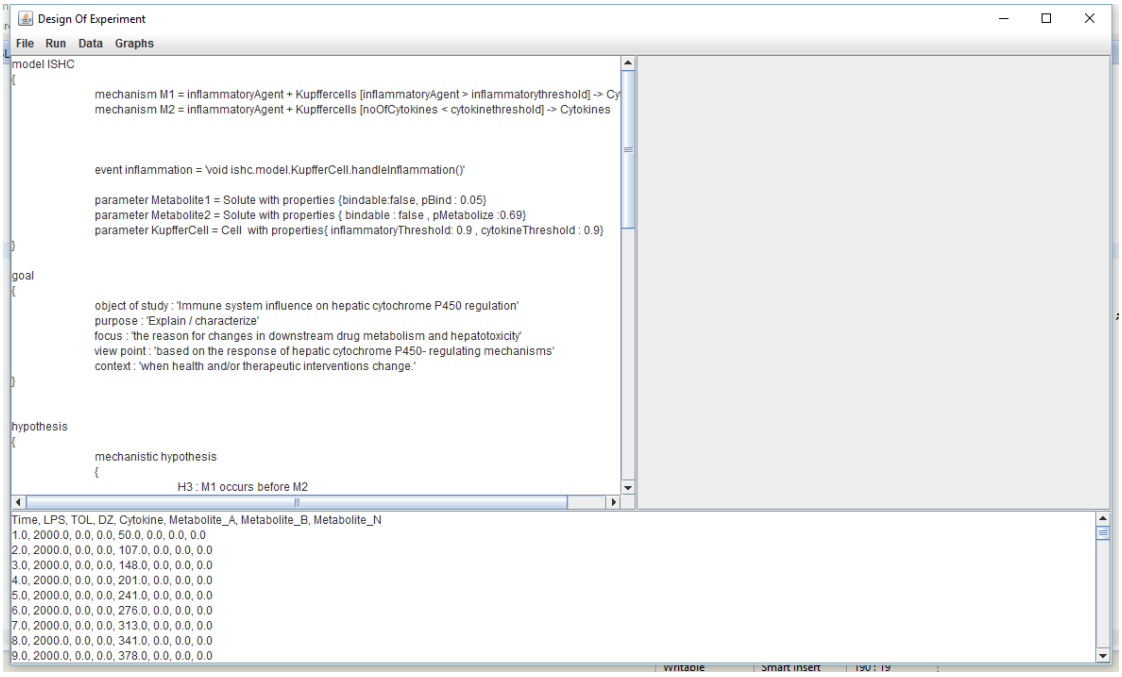


Figure 9: Result of the run is displayed

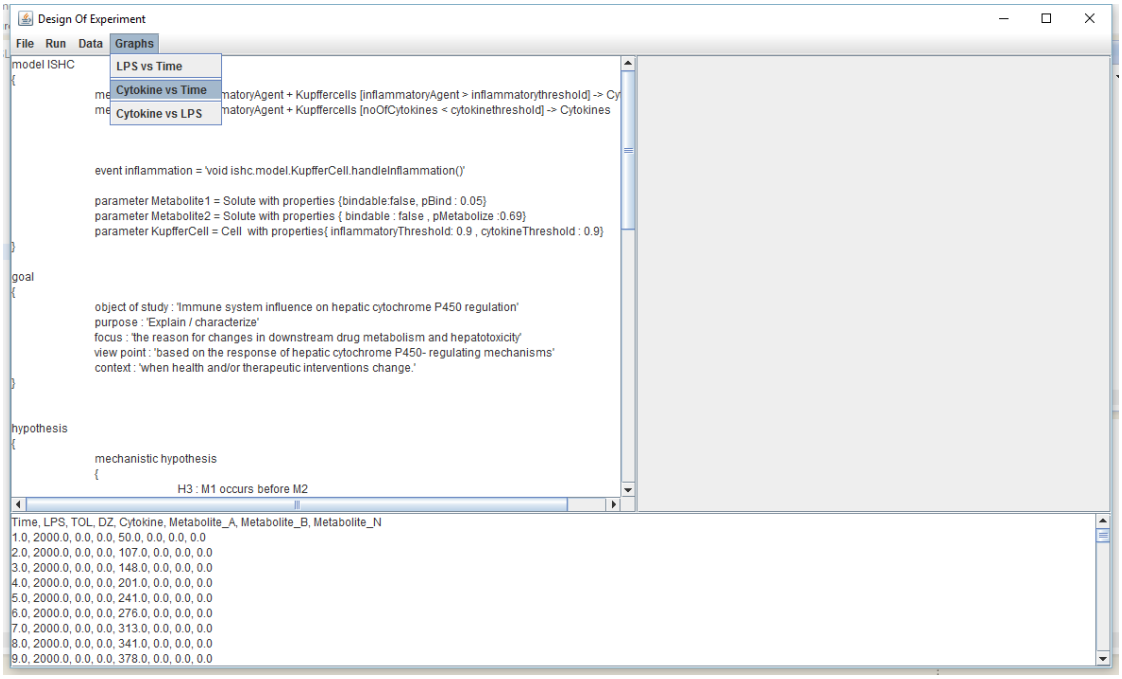


Figure 10: Graph menu shows various graphs that show the relationship between the factors of the simulation model

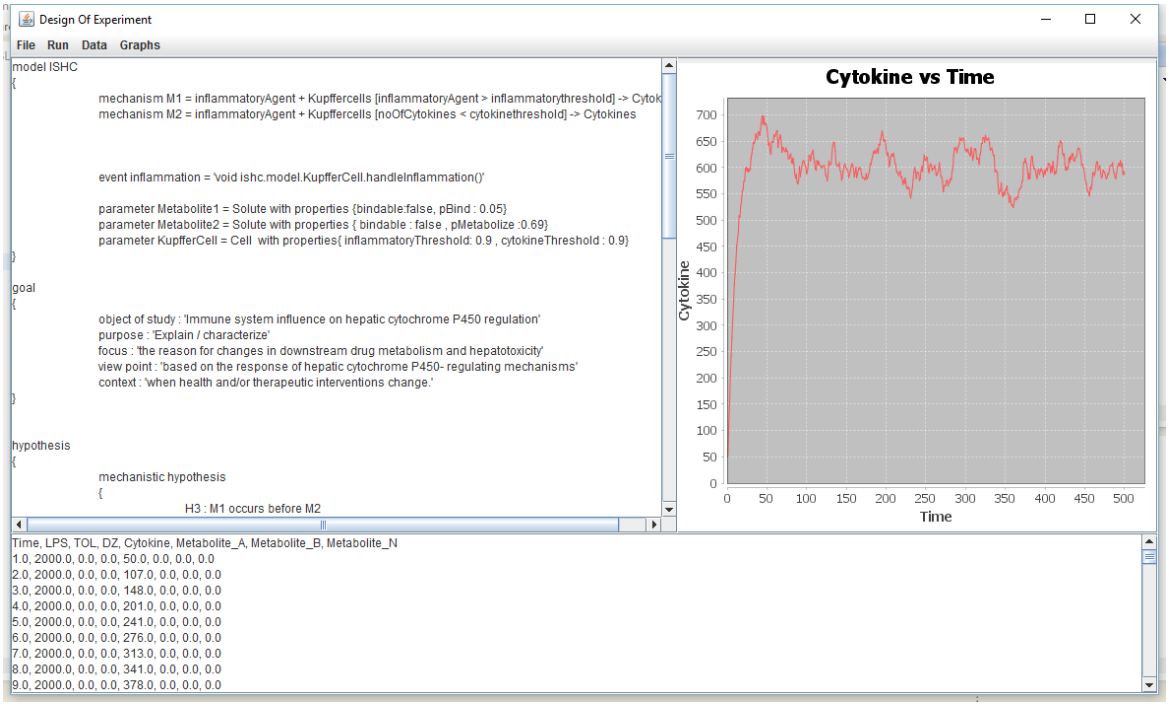


Figure 11: Graph showing the relationship between the cytokines and LPS levels

CHAPTER 6

CONCLUSION

6.1 Summary

The GHE framework has the ability to streamline execution of simulation models and verify their correctness properties through temporal attributes. By conforming to Model-Driven development practices, specifically the relation of metamodel to grammar to implementation, the project was developed in a way that makes it extensible for future work. Since the realm of experimentation and discovery is one which is constantly evolving, extensibility is a valuable attribute. By giving simulationists access to a diverse set of tools, such as experiment design strategy, factors of interest, acceptable output ranges, and temporal properties, the process of experimentation can be simplified, while hiding the details of the simulation implementation. To the user of this system, the rewards of investing in this framework are apparent in development time, convenience, and efficiency. The system does have its limitations, however. For example, since LTL can only evaluate a linear chain of events, the term “verification” of a model is somewhat misleading. For a model to be formally verified, all the states of a system should be checked for inconsistent behavior. This framework only provides support for behaviors that have been observed in models, but not those behaviors which may exist but have not been observed.

6.2 Further Topics

6.2.1 Coherence/Model Discovery

There is a great synergy to be taken advantage of between mechanistic hypotheses and temporal properties. We can take observations made from in vitro or in vivo labs as evidences to our simulation model in the form of temporal specifications. We can use these evidences in a coherence model with mechanism changes in the program, specified by a mechanistic hypothesis to see if an evidence is invalidated with the mechanism change or supports one or more evidence. One use of this coherence model is to develop an intelligent agent that can take knowledge gleaned from the model and develop new mechanism changes that support the most evidences, in an effort to develop autonomous computing. Alternatively, or in the short term, this coherence model will be useful for model discovery for a simulationist. If, for example, a simulationist introduces a new evidence to the system which is not supported by the current mechanisms, connections in the coherence model can help direct the user to a needed mechanism update that would not otherwise be known.

6.2.2 Mechanistic Hypotheses

Our future efforts will be directed towards generalizing this rule based definition of the hypotheses to capture various behavior of the model. Also our efforts will be directed towards identifying reaction scenarios for mechanistic hypotheses and associating the rules to a particular transformation process to facilitate computation. Transformation of these mechanism into computational code for

simulation is a task in progress. Generalizing the transformation process to accommodate various scenarios is also a future goal.

The challenge lies in identifying different scenarios of mechanistic hypotheses and generalizing the DSL to allow their definition. The transformation of these hypotheses to mechanisms or expected behavior in the simulation model for the purpose of computation, is a challenge. These transformations might require additional information or assumptions on the model that must be provided by the user.

6.2.3 Domain Specific Experiments

Also our efforts will be directed towards the developing domain specific experiments to verify the hypotheses. The challenge lies in mapping of the hypotheses to experimental designs. It is difficult to predict a general method to generate the designs from a hypothesis. There can be many different designs that can be used. Generation and selection of the design requires additional information that must be provided by the user and that is not necessarily related to experimental design.

BIBLIOGRAPHY

1. Alison E. Aitken and Edward T. Morgan. 2007. *Gene-specific effects of inflammatory cytokines on cytochrome P450 2C, 2B6 and 3A4 mRNA levels in human hepatocytes*.
2. Mario Bunge. 1998. *Philosophy of science: from problem to theory*, Transaction Publishers.
3. Charles Consel, Fabien Latry, Laurent Réveillere, and Pierre Cointe. 2005. A generative programming approach to developing DSL compilers. In *International Conference on Generative Programming and Component Engineering*. 29–46.
4. Andrew Crooks, Christian Castle, and Michael Batty. 2008. *Key challenges in agent-based modelling for geo-spatial simulation*,
5. Lindley Darden. 2001. Discovering mechanisms: A computational philosophy of science perspective. In *International Conference on Discovery Science*. 3–15.
6. Arie Van Deursen, Paul Klint, and Joost Visser. 2000. Domain-Specific Languages: An Annotated Bibliography. *Sigplan Not.* 35, 6 (2000), 26–36.
7. Sašo Džeroski, Pat Langley, and Ljupčo Todorovski. 2007. *Computational discovery of scientific knowledge*,
8. Roland Ewald and Adelinde M. Uhrmacher. 2014. SESSL: A domain-

specific language for simulation experiments. *ACM Trans. Model. Comput. Simul.* 24, 2 (2014), 11.

9. Dragan Gaševic, Dragan Djuric, and Vladan Devedžic. 2009. Model driven engineering. In *Model driven engineering and ontology development*. Springer, 125–155.
10. a Hallagan, B. Ward, and L.F. Perrone. 2010. An experiment automation framework for ns-3. *Proc. 3rd Int. ICST Conf. Simul. Tools Tech.* 3 (2010), 38. DOI:http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8821
11. R. Jakobovits, S.G. Soderland, R.K. Taira, and J.F. Brinkley. 2000. Requirements of a Web-based experiment management system. *Proc. AMIA Symp.* (2000), 374–8.
12. Lucas N. Joppa et al. 2013. Troubling trends in scientific software use. *Science (80-.).* 340, 6134 (2013), 814–815.
13. Jack P.C. Kleijnen, Susan M. Sanchez, Thomas W. Lucas, and Thomas M. Cioppa. 2005. State-of-the-Art Review: A User?s Guide to the Brave New World of Designing Simulation Experiments. *INFORMS J. Comput.* 17, 3 (2005), 263–289. DOI:http://dx.doi.org/10.1287/ijoc.1050.0136
14. Willi Klösgen. 1994. Exploration of Simulation Experiments by Discovery. *AAAI Tech. Report, WS-04-03* (1994).
15. Dagmar Köhn and Nicolas Le Novère. 2008. SED-ML - An XML format for the implementation of the MIASE guidelines. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* 5307 LNBI (2008), 176–190. DOI:http://dx.doi.org/10.1007/978-3-540-

88562-7_15

16. Charles W. Krueger. 1992. Software reuse. *ACM Comput. Surv.* 24, 2 (1992), 131–183.
17. Zeeya Merali. 2010. Error: Why scientific programming does not compute. *Nature* 467, 7317 (2010), 775–777. DOI:http://dx.doi.org/10.1038/467775a
18. T.C. Peachey, N.T. Diamond, D.A. Abramson, W. Sudholt, A. Michailova, and S. Amirriazi. 2008. Fractional factorial design for parameter sweep experiments using Nimrod / E. 16 (2008), 217–230. DOI:http://dx.doi.org/10.3233/SPR-2008-0250
19. L. Felipe Perrone, Christopher S. Main, and Bryan C. Ward. 2012. Safe: simulation automation framework for experiments. In *Proceedings of the Winter Simulation Conference*. 249.
20. Brenden K. Petersen, Glen E.P. Ropella, and C. Anthony Hunt. 2014. Toward modular biological models: defining analog modules based on referent physiological mechanisms. *BMC Syst. Biol.* 8 (2014), 95. DOI:http://dx.doi.org/10.1186/s12918-014-0095-1
21. Hazhir Rahmandad and John D. Sterman. 2012. Reporting guidelines for simulation-based research in social sciences. *Syst. Dyn. Rev.* 28, 4 (2012), 396–411.
22. Susan M. Sanchez, Paul J. Sánchez, and Hong Wan. 2014. Simulation experiments: better insights by design. *Proc. 2014 Summer Simul. Multiconference* (2014), 53.
23. Douglas C. Schmidt. 2006. Model-Driven Engineering. *IEEE Comput.* 39,

2 (2006), 25–31. DOI:http://dx.doi.org/10.1109/MC.2006.58

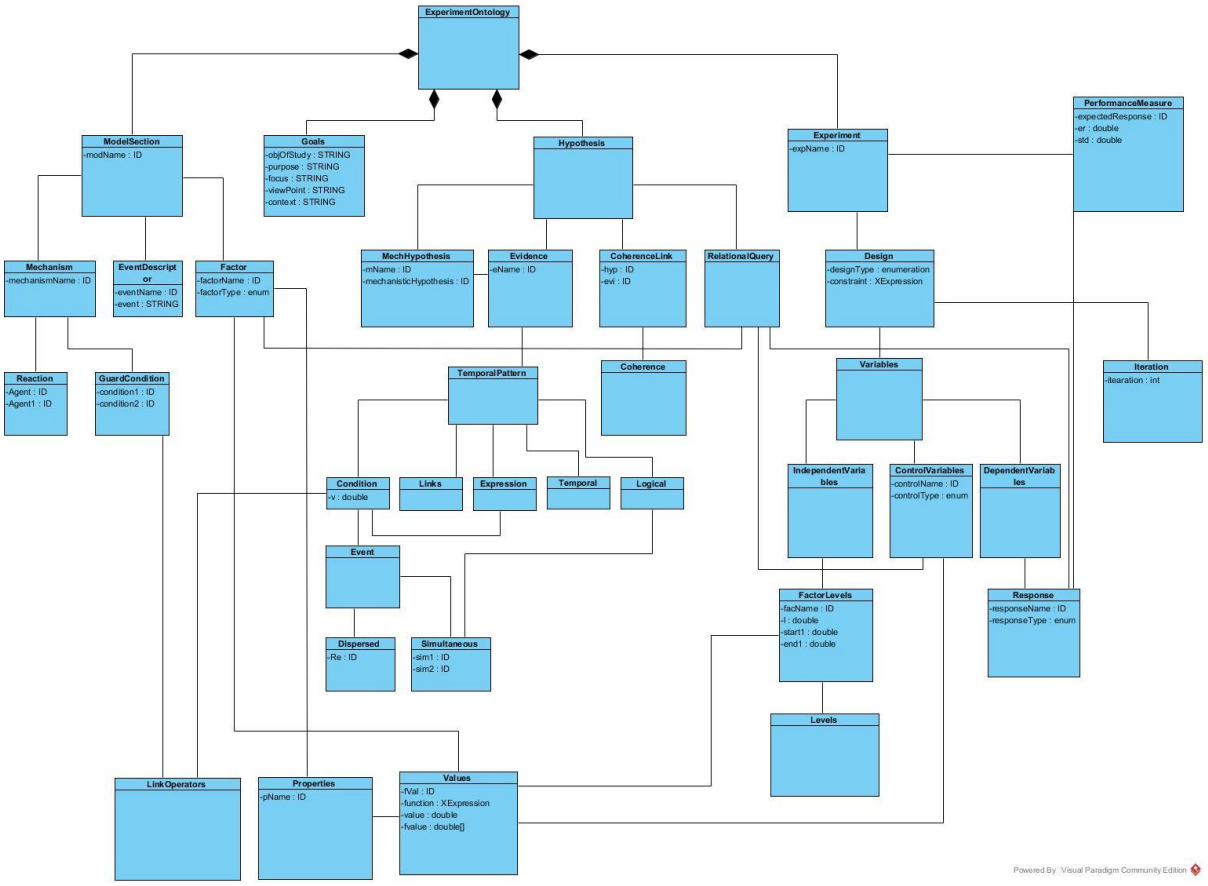
24. Johannes Schutzel, Danhua Peng, Adelinde M. Uhrmacher, and L. Felipe Perrone. 2015. Perspectives on languages for specifying simulation experiments. In *Proceedings - Winter Simulation Conference*. DOI:http://dx.doi.org/10.1109/WSC.2014.7020125
25. Gregory Sliwoski, Sandeepkumar Kothiwale, Jens Meiler, and Edward W. Lowe. 2014. Computational methods in drug discovery. *Pharmacol. Rev.* 66, 1 (2014), 334–95. DOI:http://dx.doi.org/10.1124/pr.112.007336
26. Jeffrey A. Sunman, Roy L. Hawke, Edward L. LeCluyse, and Angela D.M. Kashuba. 2004. Kupffer cell-mediated IL-2 suppression of CYP3A activity in human hepatocytes. *Drug Metab. Dispos.* 32, 3 (2004), 359–363.
27. Alejandro Teran-Somohano, Orçun Dayıbaş, Levent Yilmaz, and Alice Smith. 2014. Toward a model-driven engineering framework for reproducible simulation experiment lifecycle management. In *Proceedings of the 2014 Winter Simulation Conference*. 2726–2737.
28. Alejandro Teran-Somohano, Alice E. Smith, Joseph Ledet, Levent Yilmaz, and Halit O˘guzt¨uz¨un. 2015. A model-driven engineering approach to simulation experiment design and execution. In *Proceedings of the 2015 Winter Simulation Conference*. 2632–2643.
29. Paul Thagard. 1989. Explanatory coherence. *Behav. Brain Sci.* 12, 03 (1989), 435–467.
30. Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. 2013. *Model-driven software development: technology,*

engineering, management, John Wiley & Sons.

31. Dagmar Waltemath, Richard Adams, Daniel A. Beard, et al. 2011. Minimum information about a simulation experiment (MIASE). *PLoS Comput. Biol.* 7, 4 (2011), 5–8. DOI:http://dx.doi.org/10.1371/journal.pcbi.1001122
32. Dagmar Waltemath, Richard Adams, Frank T. Bergmann, et al. 2011. Reproducible computational biology experiments with SED-ML--the Simulation Experiment Description Markup Language. *BMC Syst. Biol.* 5, 1 (2011), 198. DOI:http://dx.doi.org/10.1186/1752-0509-5-198

APPENDIX

1. Metamodel



2. BNF grammar

<Model> ::= ExperimentOntology

<ExperimentOntology> ::= ModelSection | Goals | Hypothesis | Experiment

<ModelSection> ::= model <id> {<Mechanism> <EventDescriptor> <Factor>}

<Mechanism> ::= mechanism <id> = <Reaction> <GuardCondition> ->

<Reaction>

<Reaction> ::= <id> + <id>

<GuardCondition> ::= [<id> <LinkOperator> <id>]

<EventDescriptor> ::= event <id> = <STRING>

<Factor> ::= parameter <id> = <VariableType> <id> with values

{<Values>} properties {<properties>}

<Goals> ::= goal { object of study : <STRING> purpose : <STRING> focus

: <STRING> view point : <STRING> context : <STRING>}

<VariableType> ::= QUALITATIVE | QUANTITATIVE | CONTINUOUS |

DISCRETE | BINARY | NONBINARY

<Values> ::= <id> <XExpression> <rangeValue> <factorLevelValue>

<properties> ::= <id> : <Values>

<rangeValue> ::= INT <dot> <OptionalInt>

<factorLevelValue> ::= <rangeValue> | , <rangeValue>

<dot> ::= "." | ""

<OptionalInt> ::= INT | ""

<Hypotheses> ::= hypotheses { mechanistic

hypotheses{<MechHypotheses>} evidence{<Evidence>} coherence

model{<CoherenceLink>} relational hypotheses {<RelationalQuery>}}

<CoherenceLink> ::= <Coherence> (<id>)(<id>)

<MechHypotheses> ::= <id> : <TemporalPattern> <id>

<Evidence> ::= <id> : <TemporalPattern> activation weight :

<rangeValue>

<TemporalPattern> ::= <Sample> <Links> <Expression> <Operator>

<Links>

<Sample> ::= <Condition> | <Event>

<Condition> ::= <ComplexID> <LinkOperators> <Expression>

<rangeValue> <Condition>

<Event> ::= <Dispersed> | <Simultaneous>

<Dispersed> ::= <id> <Logical> <id> | <id>

<Operator> ::= <Temporal> | <Logical>

<Simultaneous> ::= [<id> <Logical> <id>] | [<id>]

<LinkOperators> ::= '+'|'-'|'*'|'/'|'%'|' = '|' == '|'&&'|'||'|'<'|'<='|'>'|'>='|'!'|'!='

<Expression> ::= true | false

<Links> ::= is | occurs | to | in

<Temporal> ::= precedes | between | eventually | always | before | after |

until | never | leads | absent | exists

<Logical> ::= and | or | not

<Coherence> ::= EXPLAIN | ANALOGOUS | DATA | CONTRADICT

<RelationalQuery> ::= <Query1> | <Query2> | <Query3> | <Query4> |

<Query5>

<Query1> ::= if <id> <id> is <rangeValue> <Action> then <id> is

<Response>

<Action> ::= added | removed | in the range <rangeValue> to

<rangeValue>

<Response> ::= <rangeValue> | in the range <rangeValue> to

<rangeValue>

<Query2> ::= compare <Operand> and <Operand>

<Operand> ::= <Function> | <id>

<Function> ::= MIN | MAX | EXP | INVERSE | SIN | COS | TAN |

FACTORIAL | LOG

<Query3> ::= if <QueryCondition> then <QueryResponse> where

<Levels> for <id> <id> <id> is in the range <rangeValue> to <rangeValue>

<QueryCondition ::= <id> <id> is <Level>

<Level> ::= at level <Levels> <rangeValue> <OptionalAnd> <Level> | ""

<QueryResponse> ::= <id> is <Level>

<Changes> ::= CHANGED | INCREASED | DECREASED | CONSTANT

<Levels> ::= HIGH | MEDIUM | LOW

<Experiment> ::= experiment <id> { design <Design> performance

measure is <PerformanceMeasure}

<Design> ::= {designType <DesignType> constraints <XExpression>

<Iteration> variables <Variables>}

<DesignType> ::= FULLFACTORIAL | FRACTIONALFACTORIAL |

OTHERS | ""

<Variables> ::= {<IndependentVariables> <ControlVariables>

<DependentVariables>}

<IndependentVariables> ::= independent variables {<FactorLevels>}

<ControlVariables> ::= control variables {<id> : type <VariableType> with values {<Values>}}

<DependentVariables> ::= dependent variables {<Response>}

<FactorLevels> ::= <id> are at levesl : <rangeValue> <Levels> where <Levels> is in the range <rangeValue> to <rangeValue>

<Response> ::= <id> : type <ResponseType>

<ResponseType ::= SIMPLE | COMPOSITE

<Iteration> ::= number of iterations : INT

<PerformanceMeasure> ::= {<id> = <rangeValue> +- <rangeValue>}

<OptionalAnd> ::= and | ""

<OptionalTo> ::= to | ""

<STRING> ::= "..."

3. Xtext Grammar

```
grammar org.xtext.Ontology.DOE with org.eclipse.xtext.xbase.Xbase
generate dOE "http://www.xtext.org/Ontology/DOE"
Model:
        (elements+=ExperimentOntology)*;
ExperimentOntology :
        ModelSection |Goals | Hypothesis | Experiment
;
```

```
ModelSection:
        'model' (modName = ID)
        '{'
        (mechanisms += Mechanism)*
        ((events += EventDescriptor)?)*
        (parameters += Factor)*
        '}'
;
Mechanism:
        'mechanism' (mechanismName = ID) ' = ' (LHS = Reaction) (condition =
GuardCondition)? ' -> ' (RHS = Reaction)
;
Reaction:
        (agent1 = ID) (' + ' )? (agent = ID)?
;
GuardCondition:
        gd = '[' (condition1= ID)? (link = LinkOperators)? (condition2 = ID)? ']'
;
LinkOperators:
        '+'|'-'|'*'|'/'|'%'|' = '|' == '|'&&'|'||'|'<'|'<='|'>'|'>='|'!'|'!='
;
EventDescriptor:
        'event' (eventName = ID) ' = ' (event = STRING)
```

```
;
Factor:
        'parameter' (factorName = ID) ' = ' (factorType = VariableType)?
        (factorType1 = ID)?
        'with' ('values' '{'(factorValue = Values)'}')?
        ('properties' '{'((factorProperties += properties)*)'}')?
;
enum VariableType :
        QUALITATIVE | QUANTITATIVE | CONTINOUS | DISCRETE  | BINARY |
        NONBINARY
;
properties:
        pName= ID ':' pVal = Values ','?
;
Values:
        (fVal = ID)?
        ( function = STRING)?
        ( value = rangeValue)?
        ( fvalue = factorLevelValue)?
;
terminal rangeValue         : INT ('.')? (INT)? ;
terminal factorLevelValue : rangeValue (',' rangeValue)* ;
Goals:
```

```
        'goal' '{'
        'object' 'of' 'study' ':' (objOfStudy = STRING)
        'purpose' ':' (purpose = STRING)
        'focus' ':' (focus = STRING)
        'view point' ':' (viewPoint = STRING)
        'context' ':' (context = STRING)
        '}'
;
Hypothesis :
        'hypotheses'
        '{'
        ('mechanistic' 'hypotheses'  '{'
        (mechHypothesis += MechHypothesis)*
        '}')?
        ('evidence' '{'
        (evidences += Evidence)*
        '}')?
        ('coherence' 'model' '{'
        (coherenceLinks += CoherenceLink)*
        '}')?
        ('relational' 'hypotheses' '{'
        (relHypothesis += RelationalQuery)*
        '}')?
```

'}'

;

CoherenceLink:

(coherence = Coherence) '('(hyp += ID)* ')' '('((evi += ID))*')'

;

MechHypothesis:

(mName = ID) ':'(assoMech += TemporalPattern)*

(mechanisticHypothesis = ID)?

;

Evidence:

(eName = ID) ':' (query += TemporalPattern)*

'activation' 'weight' ':' (objOfStudy = rangeValue)

;

TemporalPattern:

Condition ((l2 += Links)?)* ((exp += Expression)?)* ((op1 += Temporal |

op2 += Logical)?)* (l3 = Links)?

;

Condition:

condition = Event (lo= LinkOperators)? (e=Event)? (exp1=Expression)?

(v=rangeValue)?

;

Event:

Dispersed | Simultaneous

```
;
Simultaneous:
        '['
                sim1 = Re (log += Logical sim2 += Re)*
        ']'
;
Dispersed:
        (disp += Re)+
;
Re:
        ID ('(' (ID ID)? ')')?
;
enum Expression:
        TRUE | FALSE
;


enum Links:
        is | occurs | to | in
;
enum Temporal:
        precedes | between | eventually | always | before | after | until | never |
        leads | absent | exists
;
```

enum Logical:

and | or | not

;

enum Coherence:

EXPLAIN | ANALOGOUS | DATA | CONTRADICT

;

RelationalQuery:

Query1 | Query2 | Query3

;

Query1:

'if' (factor= ID)? (control=ID)? 'is' (x=rangeValue)?

('added')? ('removed')? ('in the range' start1=rangeValue 'to'

end1=rangeValue)?

'then' (response= ID) 'is' (y=rangeValue)? ('in the range'

start2=rangeValue 'to' end2=rangeValue)?

;

Query2:

'compare' (function1=Function)? (response1=ID)? (factor1=ID)? 'and'

(function2=Function)? (response2=ID)? (factor2=ID)?

;

enum Function:

MIN | MAX | EXP | INVERSE | SIN | COS | TAN | FACTORIAL | LOG

;

Query3:

'if' ((factor1=ID)? (control1=ID)? 'is' ('at' 'level' (Level2=Levels))?

(x2=rangeValue)? ('and')?)*

'then' ((response1=ID) 'is' ('at' 'level' (Level4=Levels))? (x4=rangeValue)?

('and')?)*

'where' ((level=Levels) 'for' (factor=ID)? (control=ID)? (response=ID)?

'is' 'in the range' (start1=rangeValue) 'to' (end1=rangeValue))*

;

enum Changes:

CHANGED | INCREASED | DECREASED | CONSTANT

;

enum Levels:

HIGH | MEDIUM | LOW

;

Experiment :

'experiment' (expName = ID)'{'

'design' (expDesign = Design)

'performance measure' 'is' (perfMeasure = PerformanceMeasure)

'}'

;

```
Design :
        '{'
        ('designType' designType = DesignType)?
        (('constraints' constraint = ID)?)*
        ( iteration = Iteration)?
        ('variables' variables = Variables)
        '}'
;
enum DesignType:
        FULLFACTORIAL | FRACTIONALFACTORIAL | OTHERS
;
 Variables:
        '{'
        ( independentVariables = IndependentVariables)
        ( controlVariables = ControlVariables)?
        ( dependentVariables = DependentVariables)
        '}'
;
IndependentVariables :
        'independent' 'variables' '{'
        ( variables +=  FactorLevels)*
        '}'
;
```

```
ControlVariables:
        'control' 'variables' '{' ((controlName = ID) ':' 'type' (controlType =
        VariableType) 'with' 'values' '{'(controlValue = Values) )* '}'
;
DependentVariables:
        'dependent' 'variables' '{'
        ( responseName = Response)*
        ;
FactorLevels:
        ((facName = ID) 'are' 'at' 'levels' ':'  (l= factorLevelValue)? ((l1 = Levels)
                'where' (l2 = Levels)
                'is' 'in the range' (start1=rangeValue) 'to' (end1=rangeValue))?)
;
Response:
        (responseName = ID) ':' 'type' (responseType = ResponseType)
        '}'
;
enum ResponseType :
        SIMPLE | COMPOSITE
;
Iteration:
        ('number' 'of' 'iterations' ':' iterations = INT)
;
```

```
PerformanceMeasure :
        '{'
                (expectedResponse = ID) '=' (er= rangeValue)(' +-')(std=
                rangeValue)
        '}'
;
```

4. Xtend Generator

```
/*
 * generated by Xtext
 */
package org.xtext.Ontology.generator
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess
import org.xtext.Ontology.dOE.Experiment
import org.xtext.Ontology.dOE.Factor
import org.xtext.Ontology.dOE.Model
import org.xtext.Ontology.dOE.ModelSection
import org.xtext.Ontology.dOE.properties
import org.xtext.Ontology.dOE.Evidence
import org.xtext.Ontology.dOE.Mechanism
```

```
/**
 * Generates code from your model files on save.
 *
 * See
https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#
code-generation
 */
class DOEGenerator implements IGenerator {

	override void doGenerate(Resource resource, IFileSystemAccess
fsa) {
		fsa.generateFile('ishc.properties',
		toISHCProperties(resource.allContents
		.filter(typeof(ModelSection)).head))
		fsa.generateFile('delivery.properties',
		toDeliveryProperties(resource.allContents
		.filter(typeof(ModelSection)).head ,
		resource.allContents.filter(typeof(Experiment)).head))
		fsa.generateFile("KupfferCell.java",
		toKupfferCell(resource.allContents
		.filter(typeof(ModelSection)).head))
		fsa.generateFile("Hepatocyte.java",
```

```
        toHepatocyte(resource.allContents
        .filter(typeof(ModelSection)).head))
}
def toISHCProperties(ModelSection m) '''
        # model parameters
        stepsPerCycle = 1
        # component parameters
        «FOR factor : m.parameters »
                «IF(factor != null)»
                        «IF(factor.factorValue != null)»
                                «IF(factor.factorValue.function != null)»
                                «factor.factorName» =
                                «factor.factorValue.function»
                                «ELSEIF(factor.factorValue.FVal
                                !=null)»
                                «factor.factorName» =
                                «factor.factorValue.FVal»
                                «ELSEIF(factor.factorValue.fvalue !=
                                null)»
                                «factor.factorName» =
                                «factor.factorValue.fvalue»
                                «ELSEIF(factor.factorValue.value !=
                                null)»
```

«factor.factorName» =

«factor.factorValue.value»

«**ENDIF**»

«**ENDIF**»

«**ENDIF**»

«**ENDFOR**»

'''

def toDeliveryProperties(ModelSection m, Experiment e)

'''

deliveryType = bolus

useContinualDoseFunction = false

repeatDose = true

infusionStopTime = 120.0

numDoses = 1

time.0 = 1.0

time.1 = 10.0

dose.0.alpha = 2000

dose.0.beta = -1

dose.0.gamma = -2

dose.0.numEntries = 7

«**var** count1 = -1»

```
«FOR factor : m.parameters »
        «IF(factor != null )»
                #«count1++»
                «IF(factor.factorProperties != null)»
                        «var count2 = 0»
                        «FOR p : factor.factorProperties»
                          «IF(!(p.PName.equals("membraneCrossing")
                        || p.PName.equals("bileRatio") ||
                        p.PName.equals("core2Rim") ||
                        p.PName.equals("metProbStart") ||
                        p.PName.equals("metProbFinish") ||
                        p.PName.equals("metabolites") ||
                        p.PName.equals("inflammatory") ||
                        p.PName.equals("pDegrade") ||
                        p.PName.equals("transportOut")))»

                                «IF(p.PVal.FVal != null)»
                        dose.0.solute.«count1».«p.PName» =
                        «p.PVal.FVal»
                                «ELSEIF(p.PVal.function !=
                                null)»
                        dose.0.solute.«count1».«p.PName» =
                        «p.PVal.function»
```

«**ELSEIF**(p.PVal.value != **null**)»

dose.0.solute.«count1».«p.PName» =

«p.PVal.value»

«**ELSEIF**(p.PVal.fvalue != **null**)»

dose.0.solute.«count1».«p.PName» =

«p.PVal.fvalue»

«**ENDIF**»

«**ELSEIF**(p.PName.equals("bileRatio") ||

p.PName.equals("core2Rim") ||

p.PName.equals("metProbStart") ||

p.PName.equals("metProbFinish") ||

p.PName.equals("inflammatory") ||

p.PName.equals("pDegrade"))»

dose.0.solute.«count1».property.«count

2».key = «p.PName»

dose.0.solute.«count1».property.«count

2».type = real

«**IF**(p.PVal.FVal != **null**)»

dose.0.solute.«count1».property.«count

2++».val = «p.PVal.FVal»

«**ELSEIF**(p.PVal.function !=

null)»

dose.0.solute.«count1».property.«count
2++».val = «p.PVal.function»
«**ELSEIF**(p.PVal.value != **null**)»
dose.0.solute.«count1».property.«count
2++».val = «p.PVal.value»
«**ELSEIF**(p.PVal.fvalue != **null**)»
dose.0.solute.«count1».property.«count
2++».val = «p.PVal.fvalue»
«**ENDIF**»
«**ELSEIF**(p.PName.equals("membraneCrossing") ||
p.PName.equals("transportOut"))»
dose.0.solute.«count1».property.«count2».key =
«p.PName»
dose.0.solute.«count1».property.«count2».type =
boolean
«**IF**(p.PVal.FVal != **null**)»
dose.0.solute.«count1».property.«count2++».val =
«p.PVal.FVal»
«**ELSEIF**(p.PVal.function != **null**)»
dose.0.solute.«count1».property.«count2++».val =
«p.PVal.function»
«**ELSEIF**(p.PVal.value != **null**)»

```
dose.0.solute.«count1».property.«count2++».val =
«p.PVal.value»
        «ELSEIF(p.PVal.fvalue != null)»
dose.0.solute.«count1».property.«count2++».val =
«p.PVal.fvalue»
        «ENDIF»
«ELSEIF(p.PName.equals("metabolites"))»
«IF(e.expDesign.variables.independentVariables.
variables != null)»
        «FOR factorLevels :
        e.expDesign.variables.independentVariables.v
        ariables»
        «IF(factorLevels.facName.equals
        ((p.PVal.function.split('-').get(0))))»
                «IF(factorLevels.l != null)»
        dose.0.solute.«count1».property.«count2».key
        = metabolites
        dose.0.solute.«count1».property.«count2».type
        = map
```

```
                                dose.0.solute.«count1».property.«count2++».v
                                al =  «p.PVal.function.split('-').get(1)» =>
                                <«factorLevels.l»>
                                «ELSEIF(factorLevels.start1 != null &&
                                factorLevels.end1 != null)»
                                dose.0.solute.«count1».property.«count2».key
                                = metabolites
                                dose.0.solute.«count1».property.«count2».type
                                = map
                               dose.0.solute.«count1».property.«count2++».val
                               =  «p.PVal.function.split('-').get(1)» =>
                               <«factorLevels.start1»,«factorLevels.end1»>
                                                «ENDIF»
                                        «ENDIF»
                                «ENDFOR»
                        «ENDIF»
                «ENDFOR»
        «ENDIF»
«ENDFOR»
        '''
```

```
def toKupfferCell(ModelSection m)'''
        package ishc.model;
        import java.lang.Math;
        import sim.field.grid.*;
        import sim.util.Bag;
        public class KupfferCell extends Cell {
        private static final org.slf4j.Logger log =
        org.slf4j.LoggerFactory.getLogger( ISHC.class );
         public KupfferCell(Culture p, ec.util.MersenneTwisterFast
        random, int x, int y) {
            super(p,random);
            setLoc(x,y);
            actionShuffler.clear();
            actionShuffler.add(new Runnable() { public void run() {
            handleInflammation(); } });
            actionShuffler.add(new Runnable() { public void run() {
            handleDegradation();}} );
          }
          BolusEntry cytokineBolusEntry = null;
          public void handleInflammation()
          {
            int numInflammatoryStimuli = 0;
            int numCytokines = 0;
```

```
for(Object o : solutes)
{
    Solute s = (Solute) o;
    if(s.hasProperty("inflammatory") &&
   ((Boolean)s.getProperty("inflammatory")))
    {
        numInflammatoryStimuli++;
    }
    if(s.type.equals("Cytokine"))
    {
        numCytokines++;
    }
   «FOR mech : m.mechanisms»
            «IF(mech.LHS.agent != null &&
            mech.LHS.agent1 != null)»
                    «IF(mech.LHS.agent.equalsIgnoreCase(
                    "Kupffercells") ||
                    mech.LHS.agent.equalsIgnoreCase("Ku
                    pffercell"))»
    if(s.type.equals("«mech.LHS.agent1»"){
                    «IF(mech.condition != null)»
                    if("«mech.condition.condition1»"
```

```
            «mech.condition.link»
        "«mech.condition.condition2»")
        «ENDIF»
        «IF(mech.RHS.agent1 != null)»
        «IF(mech.RHS.agent1.equalsIgnoreCase("Cyt
        okines") ||
        mech.RHS.agent1.equalsIgnoreCase("Cytokin
        e"))»
                    numCytokines++;
        «ELSEIF(mech.RHS.agent1.equalsIgnoreCase
        ("Inflammation") ||
        mech.RHS.agent1.equalsIgnoreCase("Inflamm
        atoryAgent"))»
                    numInflammatoryStimuli++;
        «ELSEIF(mech.RHS.agent1.equalsIgnoreCase
        ("No Inflammation")||
        mech.RHS.agent1.equalsIgnoreCase("NoInfla
        mmation"))»
                    numInflammatoryStimuli--;
        «ENDIF»
    «ENDIF»
«IF(mech.RHS.agent != null)»
```

```
                «IF(mech.RHS.agent.equals("Cytokines") ||
                mech.RHS.agent.equals("Cytokine"))»
                    numCytokines++;
                «ELSEIF(mech.RHS.agent.equalsIgnoreCase(
                "Inflammation") ||
                mech.RHS.agent.equalsIgnoreCase("Inflamma
                toryAgent"))»
                    numInflammatoryStimuli++;
                «ELSEIF(mech.RHS.agent.equalsIgnoreCase(
                "No Inflammation")||
                mech.RHS.agent.equalsIgnoreCase("NoInflam
                mation"))»
                numInflammatoryStimuli--;
                    «ENDIF»
                «ENDIF»
            «ENDIF»
        «ENDIF»
    «ENDFOR»
                    }
                }
            }
```

```
    if(numCytokines >= parent.cytokineThreshold)
    {
        return;
    }
    if(numInflammatoryStimuli >=
    parent.inflammatoryStimulusThreshold)
    {
        double probability = 1.0 - Math.exp(-
      1*(numInflammatoryStimuli -
      parent.inflammatoryStimulusThreshold) /
      parent.exponentialFactor);
        double draw = rng.nextDouble();
        if(draw <= probability)
            addCytokine();
    }
}
public Solute addCytokine()
{
      if(cytokineBolusEntry == null)
    {
        sim.util.Bag bolusEntries = ((BolusDose)
      parent.model.delivery.doses.objs[0]).solution;
        for (int i = 0; i < bolusEntries.numObjs; i++) {
```

```
                    BolusEntry be = (BolusEntry) bolusEntries.objs[i];
                    if (be.tag.equals("Cytokine")) {
                        cytokineBolusEntry = be;
                        break;
                    }
                }
            }

            //Create the Cytokine
            Solute cytokine = new Solute(cytokineBolusEntry);
            cytokine.setProperties(cytokineBolusEntry.props);
            //Add the Cytokine
            parent.solutes.add(cytokine);
                    parent.cellSpace.setObjectLocation(cytokine,
                        myX, myY);
            solutes.add(cytokine);
            return cytokine;
        }
    }
'''
def toHepatocyte(ModelSection m) '''
    package ishc.model;
```

```
import java.util.HashMap;
import java.util.LinkedHashMap;
import sim.util.Bag;
import sim.util.Double2D;
public class Hepatocyte extends Cell implements CellInfo,
EIInfo, ELInfo, MetabolismInfo {
 private static final org.slf4j.Logger log =
org.slf4j.LoggerFactory.getLogger( ISHC.class );
    HashMap<String, Double> metProbMap = new
LinkedHashMap<String,Double>();
    HashMap<String, HashMap<String,Double>>
productionMap = new
LinkedHashMap<String,HashMap<String,Double>>();
 int numEnzymesAtInit = -Integer.MAX_VALUE;
 public Hepatocyte(Culture p, ec.util.MersenneTwisterFast
random, int x, int y) {
      super(p, random);
      setLoc(x,y);
      if (parent.ei_rate > 0.0) {
           actionShuffler.add(new EIHandler((CellInfo) this,
          (BindingInfo) this, (EIInfo) this, log));
        }
        if (parent.el_rate > 0.0) {
```

```
            actionShuffler.add(new ELHandler((CellInfo) this,
          (BindingInfo) this, (ELInfo) this, log));
        }
        if (parent.useDDI) {
            actionShuffler.add(new DDIHandler((BindingInfo)
          this, rng, log, parent.pReplace));
        }
        if (parent.drRate > 0) {
            actionShuffler.add(new Runnable() { public void run()
          { handleDownRegulation(); } });
        }
    }
float ENZYME_INIT_FACTOR = 3.0f;
    public void init() {
      int min = StrictMath.round(parent.bindmin);
      int max = StrictMath.round(parent.bindmax);
      try {
        numEnzymesAtInit = rng.nextInt(max-min) + min;
      } catch (IllegalArgumentException e) {
          numEnzymesAtInit = min;
      }
for (int bNdx=0 ; bNdx<numEnzymesAtInit; bNdx++)
        binders.add(new Enzyme());
```

```
for (Object o :
    ((BolusDose)parent.model.delivery.doses.objs[0]).sol
    ution) {
  BolusEntry be = (BolusEntry) o;
  if (be.bindable) {
    double mps = (Double)be.props.get("metProbStart");
    double mpf = (Double)be.props.get("metProbFinish");
    double mp = mps + (mpf-mps)*0.5;
    metProbMap.put(be.tag, mp);
     HashMap<String,Double2D> mprodmap =
     (HashMap<String,Double2D>)
     be.props.get("metabolites");
    HashMap<String,Double> metsMap = new
     LinkedHashMap<>();
    for (java.util.Map.Entry<String,Double2D> me :
    mprodmap.entrySet()) {
      Double2D d2d = me.getValue();
      double prmin = d2d.x;
      double prmax = d2d.y;
      double prodRate = prmin + (prmax-prmin)*0.5;
      metsMap.put(me.getKey(), prodRate);
    }
```

```
      productionMap.put(be.tag,metsMap);
    }
  }
  if (!metProbMap.isEmpty()) {
       actionShuffler.add(new
      MetabolismHandler((BindingInfo) this,
      (MetabolismInfo) this, rng, log));
    }
}
public java.util.ArrayList<ishc.util.MyInt> elimQueue = null;
int rate_increment = parent.drRate;
public void handleDownRegulation()
{

    boolean thereIsACytokine = false;
      «FOR mech : m.mechanisms»
      «IF(mech.LHS.agent != null && mech.LHS.agent1 !=
      null)»
      «IF(mech.LHS.agent1.equalsIgnoreCase("Hepatocyte
      ") ||
      mech.LHS.agent.equalsIgnoreCase("Hepatocyte"))»
    «IF(mech.RHS.agent1 != null) &&
```

```
(mech.RHS.agent1.equalsIgnoreCase("Cytokines") ||
mech.RHS.agent1.equalsIgnoreCase("Cytokine"))»
                    «IF(mech.condition != null)»
                        if("«mech.condition.condition1»"
                        «mech.condition.link»
                        "«mech.condition.condition2»"){
                                thereIsACytokine = true;
                        }
                    «ENDIF»
                «ELSE»
                for(Solute s : solutes)
                {
                   if(s.type.equalsIgnoreCase("Cytokine"))
                   {
                      thereIsACytokine = true;
                      break;
                   }
                }
                «ENDIF»
                 « IF(mech.RHS.agent1 != null) &&
                 (mech.RHS.agent1.equalsIgnoreCase("Enzymes") ||
                   mech.RHS.agent1.equalsIgnoreCase("Enzyme"))»
```

```
            «IF(mech.condition != null)»
                if("«mech.condition.condition1»"
                «mech.condition.link»
                "«mech.condition.condition2»"){
                binders.add(new Enzyme());
                return;
            }
            «ENDIF»
 «ELSE»
 if(binders.size() < numEnzymesAtInit && elimQueue !=
   null && elimQueue.size() == 0 && !thereIsACytokine)
{
    if(rng.nextDouble() < parent.drReplenish)
        binders.add(new Enzyme());
    return;
}
«ENDIF»
if(elimQueue != null && elimQueue.size() > 0)
{
    int num_to_elim = (int) elimQueue.remove(0).val;
    java.util.ArrayList<Binder> to_be_removed = new
   java.util.ArrayList<Binder>();
    for(Binder b : binders)
```

```
                {
                    if(num_to_elim <= 0)
                        break;
                    if(!bound.containsKey(b)) //if unbound
                        to_be_removed.add(b);
                    if(to_be_removed.size() >= num_to_elim)
                        break;
                    «IF(mech.RHS.agent1.contains("Removed"))»
                        if(!bound.containsKey("«mech.RHS.agent1.sub
                        string(mech.RHS.agent1.indexOf("Removed"))
                        »")) //if unbound
                        to_be_removed.add(b);
                    «ENDIF»
                    «IF(mech.RHS.agent1.contains("Added"))»
                        if(!bound.containsKey("«mech.RHS.agent1.sub
                        string(mech.RHS.agent1.indexOf("Added"))»)")
                        //if unbound
                        to_be_removed.remove(b);
                        «ENDIF»
                «ENDIF»
            «ENDIF»
        «ENDFOR»
```

```
    }
  for(Binder b : to_be_removed)
      binders.remove(b);
}
Binder firstUnbound = null;
for(Binder b : binders)
{
    if(!bound.containsKey(b))
    {
        firstUnbound = b;
        break;
    }
}
if(firstUnbound == null)
    return;
  for(Solute s : solutes)
{
    if(s.type.equalsIgnoreCase("Cytokine"))
    {
        if(rng.nextDouble() < parent.drRemove)
        {
            //Add to the queue to be removed, then return
            if(elimQueue == null)
```

```
                elimQueue = new
                java.util.ArrayList<ishc.util.MyInt>();
                for(int i = 0; i < rate_increment - 1; i++)
                {
                    elimQueue.add(new ishc.util.MyInt(0));
                }
                elimQueue.add(new ishc.util.MyInt(1));
                return;
            }
        }
    }
}
//Implementations for CellInfo
public double getResources() {
    return parent.resources;
}
public int getBindmax() {return parent.bindmax;}
public int getNumEnzymesAtInit() {return
      numEnzymesAtInit; }
    //Implementations for EIInfo
public int getEIThresh() {return parent.ei_thresh;}
public double getEIRate() {return parent.ei_rate;}
public double getEIResponse() {return
```

```
        parent.ei_response_factor;}
    //Implementations for ELInfo
    public int getELThresh() {return parent.el_thresh;}
    public double getELRate() {return parent.el_rate;}
    public double getELResponse() {return
        parent.el_response_factor;}
    //Implementations for MetabolismInfo
    public HashMap<String, Double> getMetProbMap()
        {return metProbMap;}
    public HashMap<String, HashMap<String,Double>>
        getProductionMap() {return productionMap; }
    public Bag getBolusEntries() {return ((BolusDose)
        parent.model.delivery.doses.objs[0]).solution; }
    public void removeSolute(Solute s) {
       solutes.remove(s);
       parent.solutes.remove(s);
       parent.cellSpace.remove(s);
    }
    public void addSolute(Solute s) {
       parent.solutes.add(s);
       parent.cellSpace.setObjectLocation(s, myX, myY);
       solutes.add(s);
    }
```

```
                }

        '''
    }
```

5. **Reference Model**

```
model ISHC{
    mechanism M1 = inflammatoryAgent + Kupffercells [inflammatoryAgent >
    inflammatorythreshold] -> Cytokines
    mechanism M2 = inflammatoryAgent + Kupffercells [noOfCytokine >
    cytokineThreshold] -> Cytokines
    event inflammation = 'void ishc.model.KupfferCell.handleInflammation()'
    parameter LPS = Solute with properties {tag: LPS, bindable: true ,
        bolusRatio:1.0 , pExitMedia: 0.1 ,
        pExitCell: 1.0 , bindProb : 0.25 , bindCycles : 1 , numProps : 8 ,
        membraneCrossing: true, bileRatio : 0.5 , core2Rim : 0.50 ,
        metProbStart : 0.3 ,
        metProbFinish : 0.3 , metabolites: 'LPS-Metabolite_A',
        inflammatory : true , pDegrade : 0.0
    }
    parameter TOL = Solute with properties {tag: TOL, bindable:true,
        bolusRatio:0.0 , pExitMedia: 0.001 ,
        pExitCell: 1.0 , bindProb : 0.2 , bindCycles : 2 , numProps : 6 ,
```

```
        membraneCrossing: true, bileRatio : 0.5 , core2Rim : 0.50 ,
        metProbStart : 0.2 ,
        metProbFinish : 0.2 , metabolites: 'TOL-Metabolite_B'
}
parameter DZ = Solute with properties {tag: DZ, bindable:true,
        bolusRatio:0.0 , pExitMedia: 0.05 ,
        pExitCell: 1.0 , bindProb : 0.5 , bindCycles : 2 , numProps : 6 ,
        membraneCrossing: true, bileRatio : 0.5 , core2Rim : 0.50 ,
        metProbStart : 0.5 ,
        metProbFinish : 0.5 , metabolites: 'TOL-Metabolite_N'
}
parameter Cytokine = Solute with properties {tag: Cytokine,
        bindable:false, bolusRatio:0.0 , pExitMedia: 0.02 ,
        pExitCell: 0.0 , bindProb : 0.0 , bindCycles : 1 , numProps : 2 ,
        membraneCrossing: true, pDegrade : 0.1


}
parameter Metabolite_A = Solute with properties {tag: Metabolite_A,
        bindable:false, bolusRatio:0.0 , pExitMedia: 0.0 ,
        pExitCell: 0.0 , bindProb : 0.0 , bindCycles : 2 , numProps : 4 ,
        membraneCrossing: false, bileRatio : 0.5 , core2Rim : 0.50 ,
        transportOut : true
}
```

parameter Metabolite_B **=** Solute **with properties** {tag: Metabolite_B,
bindable:false, bolusRatio:0.0 , pExitMedia: 0.0 ,
pExitCell: 0.0 , bindProb : 0.0 , bindCycles : 2 , numProps : 4 ,
membraneCrossing: false, bileRatio : 0.5 , core2Rim : 0.50 ,
transportOut : true
}

parameter Metabolite_N **=** Solute **with properties** {tag:
Metabolite_N, bindable:false, bolusRatio:0.0 , pExitMedia: 0.0 ,
pExitCell: 0.0 , bindProb : 0.0 , bindCycles : 2 , numProps : 4 ,
membraneCrossing: false, bileRatio : 0.0 , core2Rim : 0.50 ,
transportOut : true
}

parameter Metabolite2 **=** Solute **with values** {0.9}

parameter forwardBias **= DISCRETE with values** {0.5}

parameter lateralBias **= DISCRETE with values** {0.5}

parameter mediaScale **= DISCRETE with values** {1000}

parameter hepDensity **= DISCRETE with values** {0.0}

parameter KCDensity **= DISCRETE with values** {0.9}

parameter bindersPerCellMin **= DISCRETE with values** {4}

parameter bindersPerCellMax **= DISCRETE with values** {8}

parameter eiThresh **= DISCRETE with values** {1}

parameter eiRate **= DISCRETE with values** {0.05}

parameter eiResponse **= DISCRETE with values** {0.25}

parameter elThresh **= DISCRETE with values** {1}

parameter elRate **= DISCRETE with values** {0.05}

parameter elResponse **= DISCRETE with values** {0.25}

parameter scale **= DISCRETE with values** {1000000}

parameter inflammatoryStimulusThreshold **= DISCRETE with values** {0}

parameter cytokineThreshold **= DISCRETE with values** {3}

parameter exponentialFactor **= DISCRETE with values** {2}

parameter drReplenish **= DISCRETE with values** {0.005}

parameter drRemove **= DISCRETE with values** {0.015}

parameter drRate **= DISCRETE with values** {30}

}

goal

{

object of study : 'Immune system influence on hepatic cytochrome P450 regulation'

purpose : 'Explain / characterize'

focus : 'the reason for changes in downstream drug metabolism and hepatotoxicity'

view point : 'based on the response of hepatic cytochrome P450-regulating mechanisms'

context : 'when health and/or therapeutic interventions change.'

}

hypotheses

{

mechanistic hypotheses

{

H1 : M1 **occurs before** M2

}

evidence

{

E1: inflammation **occurs after** inflammatoryAgent >

inflammatoryAgentThreshold

activation weight : 0.5

E2: inflammation **is absent after** cytokine < cytokineThreshold

activation weight : 0.5

}

coherence model

{

EXPLAIN (H1)(E1)

DATA (Experiment1)(E1 E2)

}

}

experiment Exp1{

design {

variables{

independent variables

{

LPS **are at levels** : **LOW where LOW is in the range** 1.0 **to** 1.0

TOL **are at levels** : **LOW where LOW is in the range** 1.0 **to** 1.0

DZ **are at levels** : **LOW where LOW is in the range** 1.0 **to** 1.0

}

dependent variables

{

cytokines : **type SIMPLE**

}

}

}

performance measure is

{

cytokines= 500 **+-**10

}

}

6. Generated Artifacts

i. ishc.properties

model parameters

stepsPerCycle = 1

component parameters

Metabolite2 = 0.9

forwardBias = 0.5

lateralBias = 0.5

mediaScale = 1000

hepDensity = 0.0

KCDensity = 0.9

bindersPerCellMin = 4

bindersPerCellMax = 8

eiThresh = 1

eiRate = 0.05

eiResponse = 0.25

elThresh = 1

elRate = 0.05

elResponse = 0.25

scale = 1000000

inflammatoryStimulusThreshold = 0

cytokineThreshold = 3

exponentialFactor = 2

drReplenish = 0.005

drRemove = 0.015

drRate = 30

ii. delivery.properties

deliveryType = bolus

useContinualDoseFunction = false

repeatDose = true

infusionStopTime = 120.0

numDoses = 1

time.0 = 1.0

time.1 = 10.0

dose.0.alpha = 2000

dose.0.beta = -1

dose.0.gamma = -2

dose.0.numEntries = 7

#-1

dose.0.solute.0.tag = LPS

dose.0.solute.0.bindable = true

dose.0.solute.0.bolusRatio = 1.0

dose.0.solute.0.pExitMedia = 0.1

```
dose.0.solute.0.pExitCell = 1.0
dose.0.solute.0.bindProb = 0.25
dose.0.solute.0.bindCycles = 1
dose.0.solute.0.numProps = 8
dose.0.solute.0.property.0.key = membraneCrossing
dose.0.solute.0.property.0.type = boolean
dose.0.solute.0.property.0.val =  true
dose.0.solute.0.property.1.key = bileRatio
dose.0.solute.0.property.1.type = real
dose.0.solute.0.property.1.val =  0.5
dose.0.solute.0.property.2.key = core2Rim
dose.0.solute.0.property.2.type = real
dose.0.solute.0.property.2.val =  0.50
dose.0.solute.0.property.3.key = metProbStart
dose.0.solute.0.property.3.type = real
dose.0.solute.0.property.3.val =  0.3
dose.0.solute.0.property.4.key = metProbFinish
dose.0.solute.0.property.4.type = real
dose.0.solute.0.property.4.val =  0.3
dose.0.solute.0.property.5.key = metabolites
dose.0.solute.0.property.5.type = map
dose.0.solute.0.property.5.val =  Metabolite_A => <1.0,1.0>
dose.0.solute.0.property.6.key = inflammatory
```

dose.0.solute.0.property.6.type = real

dose.0.solute.0.property.6.val = true

dose.0.solute.0.property.7.key = pDegrade

dose.0.solute.0.property.7.type = real

dose.0.solute.0.property.7.val = 0.0

#0

dose.0.solute.1.tag = TOL

dose.0.solute.1.bindable = true

dose.0.solute.1.bolusRatio = 0.0

dose.0.solute.1.pExitMedia = 0.001

dose.0.solute.1.pExitCell = 1.0

dose.0.solute.1.bindProb = 0.2

dose.0.solute.1.bindCycles = 2

dose.0.solute.1.numProps = 6

dose.0.solute.1.property.0.key = membraneCrossing

dose.0.solute.1.property.0.type = boolean

dose.0.solute.1.property.0.val = true

dose.0.solute.1.property.1.key = bileRatio

dose.0.solute.1.property.1.type = real

dose.0.solute.1.property.1.val = 0.5

dose.0.solute.1.property.2.key = core2Rim

dose.0.solute.1.property.2.type = real

dose.0.solute.1.property.2.val = 0.50

dose.0.solute.1.property.3.key = metProbStart

dose.0.solute.1.property.3.type = real

dose.0.solute.1.property.3.val = 0.2

dose.0.solute.1.property.4.key = metProbFinish

dose.0.solute.1.property.4.type = real

dose.0.solute.1.property.4.val = 0.2

dose.0.solute.1.property.5.key = metabolites

dose.0.solute.1.property.5.type = map

dose.0.solute.1.property.5.val = Metabolite_B => <1.0,1.0>

#1

dose.0.solute.2.tag = DZ

dose.0.solute.2.bindable = true

dose.0.solute.2.bolusRatio = 0.0

dose.0.solute.2.pExitMedia = 0.05

dose.0.solute.2.pExitCell = 1.0

dose.0.solute.2.bindProb = 0.5

dose.0.solute.2.bindCycles = 2

dose.0.solute.2.numProps = 6

dose.0.solute.2.property.0.key = membraneCrossing

dose.0.solute.2.property.0.type = boolean

dose.0.solute.2.property.0.val = true

dose.0.solute.2.property.1.key = bileRatio

dose.0.solute.2.property.1.type = real

dose.0.solute.2.property.1.val = 0.5

dose.0.solute.2.property.2.key = core2Rim

dose.0.solute.2.property.2.type = real

dose.0.solute.2.property.2.val = 0.50

dose.0.solute.2.property.3.key = metProbStart

dose.0.solute.2.property.3.type = real

dose.0.solute.2.property.3.val = 0.5

dose.0.solute.2.property.4.key = metProbFinish

dose.0.solute.2.property.4.type = real

dose.0.solute.2.property.4.val = 0.5

dose.0.solute.2.property.5.key = metabolites

dose.0.solute.2.property.5.type = map

dose.0.solute.2.property.5.val = Metabolite_N => <1.0,1.0>

#2

dose.0.solute.3.tag = Cytokine

dose.0.solute.3.bindable = false

dose.0.solute.3.bolusRatio = 0.0

dose.0.solute.3.pExitMedia = 0.02

dose.0.solute.3.pExitCell = 0.0

dose.0.solute.3.bindProb = 0.0

dose.0.solute.3.bindCycles = 1

dose.0.solute.3.numProps = 2

dose.0.solute.3.property.0.key = membraneCrossing

dose.0.solute.3.property.0.type = boolean

dose.0.solute.3.property.0.val = true

dose.0.solute.3.property.1.key = pDegrade

dose.0.solute.3.property.1.type = real

dose.0.solute.3.property.1.val = 0.1

#3

dose.0.solute.4.tag = Metabolite_A

dose.0.solute.4.bindable = false

dose.0.solute.4.bolusRatio = 0.0

dose.0.solute.4.pExitMedia = 0.0

dose.0.solute.4.pExitCell = 0.0

dose.0.solute.4.bindProb = 0.0

dose.0.solute.4.bindCycles = 2

dose.0.solute.4.numProps = 4

dose.0.solute.4.property.0.key = membraneCrossing

dose.0.solute.4.property.0.type = boolean

dose.0.solute.4.property.0.val = false

dose.0.solute.4.property.1.key = bileRatio

dose.0.solute.4.property.1.type = real

dose.0.solute.4.property.1.val = 0.5

dose.0.solute.4.property.2.key = core2Rim

dose.0.solute.4.property.2.type = real

dose.0.solute.4.property.2.val = 0.50

dose.0.solute.4.property.3.key = transportOut

dose.0.solute.4.property.3.type = boolean

dose.0.solute.4.property.3.val = true

#4

dose.0.solute.5.tag = Metabolite_B

dose.0.solute.5.bindable = false

dose.0.solute.5.bolusRatio = 0.0

dose.0.solute.5.pExitMedia = 0.0

dose.0.solute.5.pExitCell = 0.0

dose.0.solute.5.bindProb = 0.0

dose.0.solute.5.bindCycles = 2

dose.0.solute.5.numProps = 4

dose.0.solute.5.property.0.key = membraneCrossing

dose.0.solute.5.property.0.type = boolean

dose.0.solute.5.property.0.val = false

dose.0.solute.5.property.1.key = bileRatio

dose.0.solute.5.property.1.type = real

dose.0.solute.5.property.1.val = 0.5

dose.0.solute.5.property.2.key = core2Rim

dose.0.solute.5.property.2.type = real

dose.0.solute.5.property.2.val = 0.50

dose.0.solute.5.property.3.key = transportOut

dose.0.solute.5.property.3.type = boolean

dose.0.solute.5.property.3.val = true

#5

dose.0.solute.6.tag = Metabolite_N

dose.0.solute.6.bindable = false

dose.0.solute.6.bolusRatio = 0.0

dose.0.solute.6.pExitMedia = 0.0

dose.0.solute.6.pExitCell = 0.0

dose.0.solute.6.bindProb = 0.0

dose.0.solute.6.bindCycles = 2

dose.0.solute.6.numProps = 4

dose.0.solute.6.property.0.key = membraneCrossing

dose.0.solute.6.property.0.type = boolean

dose.0.solute.6.property.0.val = false

dose.0.solute.6.property.1.key = bileRatio

dose.0.solute.6.property.1.type = real

dose.0.solute.6.property.1.val = 0.0

dose.0.solute.6.property.2.key = core2Rim

dose.0.solute.6.property.2.type = real

dose.0.solute.6.property.2.val = 0.50

dose.0.solute.6.property.3.key = transportOut

dose.0.solute.6.property.3.type = boolean

dose.0.solute.6.property.3.val = false

iii. KupfferCell.java

```
package ishc.model;
import java.lang.Math;
import sim.field.grid.*;
import sim.util.Bag;
public class KupfferCell extends Cell {
   private static final org.slf4j.Logger log =
         org.slf4j.LoggerFactory.getLogger( ISHC.class );
   public KupfferCell(Culture p, ec.util.MersenneTwisterFast
         random, int x, int y) {
      super(p,random);
      setLoc(x,y);
      actionShuffler.clear();
      actionShuffler.add(new Runnable() { public void run() {
         handleInflammation(); } });
      actionShuffler.add(new Runnable() { public void run() {
         handleDegradation();}} );
   }
   BolusEntry cytokineBolusEntry = null;
   public void handleInflammation()
   {
      int numInflammatoryStimuli = 0;
      int numCytokines = 0;
```

```
for(Object o : solutes)
{
    Solute s = (Solute) o;
    if(s.hasProperty("inflammatory") &&
   ((Boolean)s.getProperty("inflammatory")))
    {
        numInflammatoryStimuli++;
    }
    if(s.type.equals("Cytokine"))
    {
        numCytokines++;
    }
    if(s.type.equals("inflammatoryAgent"){
            if("inflammatoryAgent" > "inflammatorythreshold")
            numCytokines++;
              }
   if(s.type.equals("inflammatoryAgent"){
            if("noOfCytokine" > "cytokineThreshold")
                    numCytokines++;
            }
    }
}
```

```
    if(numCytokines >= parent.cytokineThreshold)
    {
        return;
    }
    if(numInflammatoryStimuli >=
      parent.inflammatoryStimulusThreshold)
    {
        double probability = 1.0 - Math.exp(-
      1*(numInflammatoryStimuli –
      parent.inflammatoryStimulusThreshold) /
                parent.exponentialFactor);
        double draw = rng.nextDouble();
        if(draw <= probability)
            addCytokine();
    }
}
public Solute addCytokine()
{
      if(cytokineBolusEntry == null)
    {
        sim.util.Bag bolusEntries = ((BolusDose)
      parent.model.delivery.doses.objs[0]).solution;
```

```
            for (int i = 0; i < bolusEntries.numObjs; i++) {
                BolusEntry be = (BolusEntry) bolusEntries.objs[i];
                if (be.tag.equals("Cytokine")) {
                    cytokineBolusEntry = be;
                    break;
                }
            }
        }
        //Create the Cytokine
        Solute cytokine = new Solute(cytokineBolusEntry);
        cytokine.setProperties(cytokineBolusEntry.props);
          //Add the Cytokine
        parent.solutes.add(cytokine);
        parent.cellSpace.setObjectLocation(cytokine, myX, myY);
        solutes.add(cytokine);
        return cytokine;
    }
}
```

7. Reference Implementation

https://github.com/szc0098/Reference-ImplementationI-SHC-model