# Exploring Model Driven Engineering from Behavioral Models

Muideen Adesola Ajagbe

**Masters By Research**

University of York
Computer Science

December 20, 2016

**Abstract**

# Chapter 1

# Introduction

This report concerns the application of model driven engineering (MDE) techniques and tools to support creation of simulator code. The work is part of an initiative to provide software engineering support for the modeling and development approach used in immune systems simulation by the York Computational Immunology Lab (YCIL)[1]

Model-Driven Engineering (MDE) is a software development approach that seeks to use models as first-class engineering artefacts in the software development lifecycle. It improve systems productivity, maintainability and reusability by using models to manipulate systems in order to understand their emergent behavior through generation of OO code. Also, many widely-used MDE techniques and tools such as EMF and Epsilon are utilized towards this end, thereby enabling large complex systems model manipulations.

Software engineering support for modeling of immune systems simulations helps with the development of fine-grained UML models from the resources provided by domain experts. This supports enables the creation of a detailed representation of different level of abstraction observed in a software system.

YCIL thesis by [1] presents different domain of an EAE immune system developed with UML behavioral diagrams which are used as our domain of interest. Our review being done on this papers explore these domain of interest and developed different approach to their modeling in other to generate Object Oriented (OO) simulator code which can be analyzed in other to understand their behaviors.

## 1.1 Overview of Model and Model Driven Engineering

Model is an abstraction used in understanding a concept. In MDE, models are represented in well-defined modeling languages. The definition of the modeling language is specified by a metamodel. A good model is said to conform to its

[1] York Computation Immunology Group: https://www.york.ac.uk/computationalimmunology/publications/

metamodel. A metamodel is itself a model, and is usually defined in a metamodelling language such as MOF or Ecore.

By defining and instantiating metamodelling approach, software engineers are able to develop well-defined domain specific languages (DSL), whilst model management tools that operate on well defined languages provide support for construction, manipulation and validation of models.

MDE-supported development typically uses a structure diagram (a UML class diagram or similar) as a basis for generating OO code. However, MDE has not generally been used to support systems modelled primarily through their behavior - a major aspect of our work.

## 1.2 Research simulations: YCIL immune systems studies

In this project, work from existing case studies created by the YCIL group are explored. This work uses behavioral models such as activity diagrams and state diagrams to create a domain model which biologists can understand and validate. These diagrams are typically created using UML with slight variations. As this behavior models are the motivation for our project, the models are currently used for validation of the domain model, and as a guide to implementation, but the code is not directly derived from the models.

The work in this report is based on the study of a murine autoimmune disease called experimental autoimmune encephalomyelitis (EAE) by [1]. Our work is focused on the EAE systems domain and how the domain models can be simulated and modelled in other to study the emergent observed in them. More so, the domain models expressed in [1] are mostly represented with behavior model like activity diagram and state diagrams showing the role of each activity and state in the EAE immune system. The need to transform this behavior diagram to structure diagram is to derive classes that detail the biological behavior observed in the immune system.

Read's thesis [1] creates his simulation using the CoSMoS [2] approach where the domain, domain model, platform model, simulation platform and result model are developed and enabled for a system as an effective process towards simulations. This process enables the use of simulation for investigations there by performing rigorous modeling activities. A process like these enables the development of computational techniques for the exploitation of many complex systems among which are validation and transformation. The benefits of using a CoSMoS-like approach on case studies such as this one include:

- The capturing and merging of vast amount of data from different sources which can be used to developing a system-level synopsis of the behavior role of the data.

[2] York Centre for Complex Systems Analysis :https://www.york.ac.uk/yccsa/research/cosmos/

- Simulation provide a platform for the formulation and valuation of hypotheses concerning complex systems operation. Also, wet-lab experimentation can be conducted easily with preliminary investigations pointing to specific area of systems being studied.

## 1.3 Motivation and Research Hypothesis

The YCIL case studies [1, 2, 3] uses a common set of behavioral modeling approaches (based on UML activity diagram and state diagrams) to handcraft code for simulation. The act of hand-crafting code is error prone, since it is difficult to provide validation that the models have been correctly interpreted in the code. MDE approaches may provide a way to rigorously transform models to code which is more fit-for-purpose.

This project investigates the hypothesis that metamodelling and model transformation can be used to provide an automatable approach to creating simulation code from the behavioral models used by YCIL projects.

To investigate this hypothesis, the report presents two approaches to automation. The first (naive) approach builds on published work that shows how a class model can be derived from an activity diagram alone. The second approach, which is more relevant to the large, complex systems addressed by YCIL, uses all the domain models to create a class model suitable for OO code generation.

Finally, the work presents a detailed review of behavioral transformation of UML behavior diagrams to structure diagrams. The effective current-state of the art tools and techniques used in MDE, for the naive and second approach, showing their benefits and limitation is discussed. It is imperative to note that our proposed solution is aimed at offering suitable approach for different scenarios experienced in different domain models.

## 1.4 Thesis Structure

Chapter 2 provides an overview of EAE immunology disease as a complex systems with the sections presenting the low level, top level of the systems model and introducing the reader to the domain of the model and the roles of the classes. Section 2.2 reviews the CoSMoS framework used in the modeling and how it correlates to the individual domain of the system. Section 2.3 provides an overview of MDE and how it helps in the development of an approach towards our transformation of models from one level of abstraction to the other.

Chapter 3 presents the hypothesis this research aims to prove and the breakdown of the approach being used to build on Mark Reads work. Research objectives and the scope the work is restricted to is detailed.

Chapter 4 details the software engineering development process used in building the model, the different models used in each level of abstraction .

Chapter 5 analyzes the object, validation and transformation languages used in building test cases for the transformed model. Strategy used for evaluating the system is presented, automation effort, the result and their significance to the modeling and simulation of EAE model is detailed. Alternative solutions that can be used as part of this work and process for evaluating the whole system is discussed.

Chapter 6 blends a summary of the understanding acquired from the research to a conclusion of the direction of this research in future. It also identify its contribution to the field of software-engineering and model-driven software development.

# Chapter 2

# Background

This chapter presents an introduction with concepts to Model Driven Engineering (MDE). It also describe YCCSA CoSMoS framework for complex systems. A detailed context of MDE is needed in other to understand the source of model validation and transformation. With a model being an abstraction of a system, they are necessary artefacts needed to understand the nitty-gritty of a system under study. In MDE, a principled process to software engineering where models are used throughout the engineering process is discussed. Section 2.1 details the fundamental concepts, terminologies used in MDE. Section 2.2 reviews the MDE tools used in our work. Section 2.3 briefly describe the benefits of using MDE. Section 2.4 focuses on the CoSMoS approach and how they aid model development.

## 2.1 MDE Concepts and Terminologies

MDE are used to developed and manipulate objects through software engineering practices in other to understand patterns and emergent behaviors in the objects [4]. MDE deploys model management operations on models, metamodels and other artefacts to achieve this. This section details the activities involved in MDE.

### 2.1.1 Models and Metamodels

MDE as a method to software systems developments allows model to be used as a first-class artefacts during development processes. "Model provides the representation of simplified system in a well-defined language" [5]. According to [6], these models are "the descriptions of phenomena of interest" mostly expressed in several general-purpose and specific languages. Also, a "model allows concepts to be shown, wherein, an abstraction of irrelevant details from reality is made possible" [7].

Whilst there are many meanings given to models, it is essentially the actual representation of artefacts in the real-world [8]. Model representing the real-world are either called domain or system of interest. The exploration of this models produces automated tools for analysis, validation, transformation and other intended

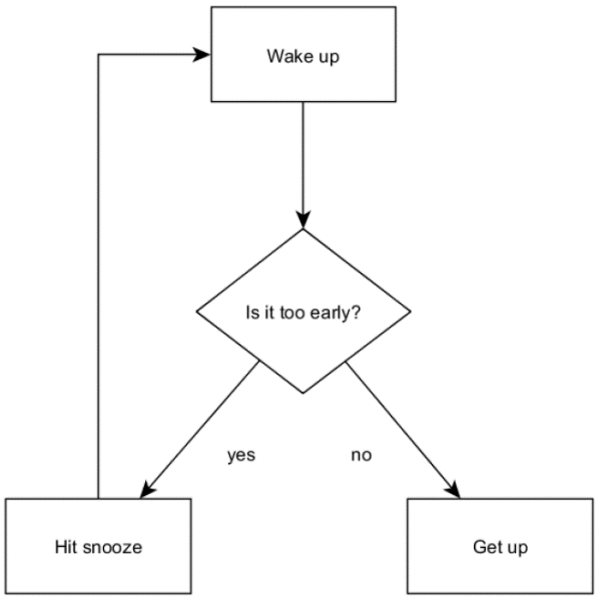


Figure 2.1: A model of a real world scenario by [Dimitris Kolovos].

purposes.

In MDE, descriptions of interest can be easily constructed and manipulated by powerful automated model management tools [9]. Model management processes allow model to be amenable to automation when they are defined in modeling languages i.e. metamodels [10].

Metamodels describes classes and the rules pertinent to a group of models [9]. The metamodels are models themselves and they conform to their instances called metametamodel [9]. Models on the other hand, are also particular instance of classes and their inherent rules. The instantiation or development of metamodel enables operation to be managed and applied to models. Metamodel creates a type system for models which allow their properties to be substantiated and validated. They aid model swapping and, therefore, interoperability between modeling artefacts and tools. This section presents important areas of MDE, model management approaches, their importance and how they relate to our work.

### 2.1.2 Modelling Languages and Semantics

Models adheres to the rigid set of rules and syntax established in their modeling language therefore they depend on metamodels [11]. MDE buttress the development of both the abstract syntax and concrete syntax with semantics for a model. These two syntax represents the metamodels and their editors respectively. More so, the metamodels are used in either a domain- specific language (DSL) or general purpose language (GPL) to archetype a model, depending on the level of abstractions involved in the domain of interest [11]. DSL and GPL are deployed for a fit-for-purpose modelling of our work.

**General Purpose Language** - The general purpose language used for our

work include UML language widely used by modellers to model the abstraction experienced in a system, from different levels. This allow the development of variety of models to study both the behavior and structure of our system. Specifically, our domain of interest used in Reads thesis [1] are modelled with UML with slight biological variations.

**Domain-Specific Language** - In MDE, DSL are tailored around a particular domain under manipulation. For our work, languages such as EOL[1], EVL[2], ETL[3] are used to query, validate and transform our models respectively. While these languages scope are limited to the domain being explored, they provide concise solution to the same level of abstraction as the problem domain, by being less suceptible to portability [12].

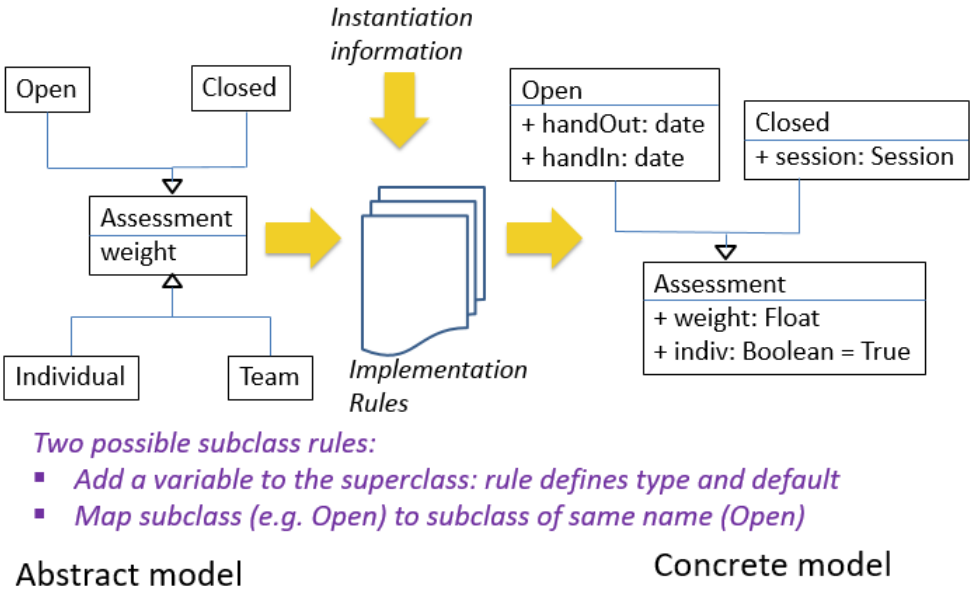


Figure 2.2: Mapping concepts: abstract to concrete UML concepts by [Fiona et al] .

It is noteworthy to understand that the abstract syntax describes objects defined in a language such as classes, packages, interfaces and datatypes [11]. Abstract syntax concepts are representations of the things to be modelled. Concrete syntax is utilized by model to represent its modelling concept. The concrete syntax provides detailed representations for constructing models that is in accordance to their language. Also, the semantics gives meaning to the modelling concepts used in a particular domain. Semantics dictate what can be done to the model. In MDE, these three objects are used together to depict a modelling language[13].

[1]Epsilon Object Language : http://www.eclipse.org/epsilon/doc/eol/
[2]Epsilon Validation Language : http://www.eclipse.org/epsilon/doc/evl/
[3]Epsilon Transformation Language : http://www.eclipse.org/epsilon/doc/eol/

### 2.1.3 MOF and Ecore: A Metamodelling Language

Software engineering support for MDE enables the development and reusability of metamodels. Several languages support the instantiating of these metamodels, thereby, making model conforms to them. For model interoperability, all these languages provides a common basis for defining models and metamodels. A metamodel language by EMF[4] called Ecore which is used for describing and supporting models is utilized in our work. Also, Object Management Group (OMG)[5] presents a standard language for specifying metamodels called the MetaObject Facility (MOF)[6].

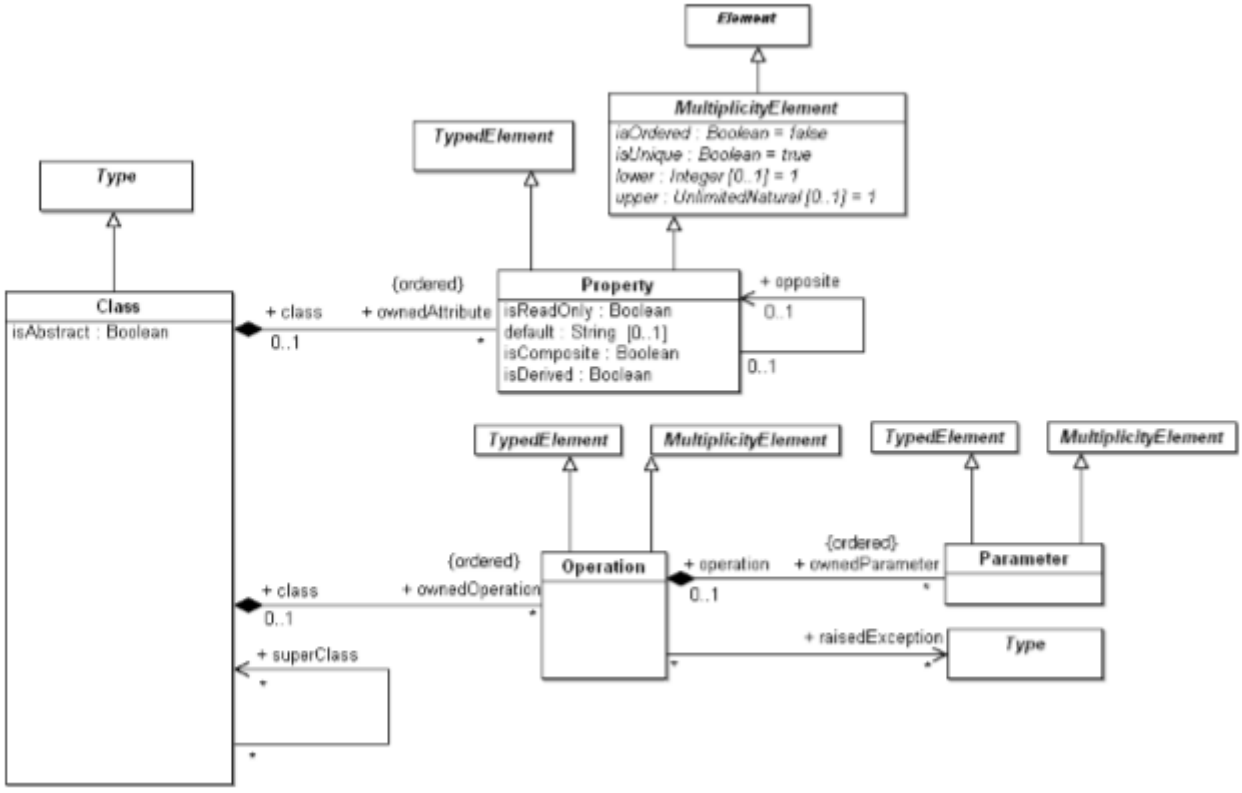


Figure 2.3: An excerpt of the UML metamodel defined in MOF, from [OMG2007a].

MOF is a modelling language for defining metamodels hence it is sometimes referred to as a metamodeling language. It is an approach to specifying modelling language with a unified metamodeling language. MOF guarantees consistency in the approach towards description of model development. Also, it supports the development of maneuverable MDE tools which can be used for model management. Ecore in the same vein, by influencing its constructs, allows the expression of models.

Using MOF and Ecore as a modelling language enables many aspects of a

[4]Eclipse Modeling Framework : https://www.eclipse.org/modeling/emf/
[5]Object Management group : http://www.omg.org/spec/
[6]Meta Object Facility : http://www.omg.org/spec/MOF/2.5.1/

metamodel to be defined. Specifically, the concrete syntax discussed in Subsection 2.1.2 which can come in form of UML structure diagrams is mostly used for developing MOF. In MDE, Ecore and MOF is needed to help bridge the gap between different metamodeling language used in defining a modelling language. A challenge and misunderstanding arises in identifying similarities when different metamodeling languages are used for a specific model [14].

### 2.1.4 Model Management

The utilization of MDE enables models to be manipulated and managed so as to produce software (models, codes and other artefacts) leading to the better understanding their emergent behaviors. The act of model management gives room for collections of operations to be accessed in manipulating models. This thesis uses model management operation to manipulate behavior models presented in Read's thesis [1] so as to underpin biological behaviors observed in the domain of interest. Model management operations such as model transformation and validation are used in our work to produce the model from our domain.

#### Model Transformation

To generate useful and fit-for-purpose model which can be used to generate code and other documents, we need to transform our models to reflect a structural representation of our domain. Model transformation is a development operation where software artefact are derived from another presented artefact. Model transformation is widely characterized as the cradle of MDE [**?**, 10 jimi] Other model management operations such as refactoring, merging, code generation would not be applicable without a concrete transformation [jimi 10, 41].

Essentially, the transformation of models are often specified through transformation rules in other to enhance their quality, recognise emergent patterns and automate software evolutions among many other attributes [jimi 42]. Model transformations have many types among which are model to model (mostly between modeling languages)and model to text (mostly between model and a textual objects).

**Model to Model (M2M)**

This is an approach to model transformation where models are derived from other model. M2M when automated helps reduce engineering cost of complex systems instead of transformation between pairs of interdependent models [15]. Using M2M approach, input model (also called source model) conforming to a metamodel, is transformed by following a set of transformation rules, in a transformation language to an output model (called target model) conforming to another metamodel. M2M transformation languages are detailed below;

- **Declarative M2M** transformation languages defines a relationship between a source and targets model as expressed earlier. In relation to our work,

```
// Transforms a class into a table and
// a primary key column
rule Class2Table
  transform c : OO!Class
  to t : DB!Table, pk : DB!Column {

  t.name = c.name;
  t.database = db;
```

Figure 2.4: ETL Model to Model transformation from a class to a table from Epsilon ETL

the behavior model presented in Read's thesis [1] is the source model while the target is the structure model our work developed. The limitation of this approach is its inability to produce fine grained rule scheduling for executable transformations. Example of Declarative M2M includes QVT-relations from OMG.

- **Imperative M2M** transformation languages defines a series of steps needed for transformation of models from the source to the target model. However, it enables exclusive control of transformation rules. The limitations to this approach is the difficulty in writing and maintaining the language [16]. Example of Imperative M2M includes QVT-operational from OMG.
- **Hybrid M2M** transformation seeks to combine declarative and imperative M2M by providing both implicit and explicit rule scheduling. Our work made use of a hybrid M2M languages in order to handle complex transformation scenarios observed in our model. The reason for adopting this language is their ability to specify model transformation as in contrast to declarative or imperative M2M languages. Example of Hybrid M2M includes the ETL which will be further discussed in subsequent chapters.

### Model to Text (M2T)

This is an approach to model transformation where models can be serialized or interchanged, code and other textual artefacts can be generated, and model can be visualized and explored. It is used to transform behavior models into structure (Object-oriented code) and unstructured artefacts (textual documentations). Using M2T approach, input models that conforms to a metamodel is transformed adhering to a transformation rule of a transformation language to a text file or a code. Examples of M2T transformation languages include JET, Xpand, Acceleo and EGL.

The primary objective of our work is to understand the emergent behavior and patterns observed in the domain of interest from Read's thesis [1]. This gives basis for the is need to transform the provided models to a textual artefacts or OO code which can be deciphered. By using EGL, our object oriented model conforming to a metamodel is transformed to an output which is a OO Java code. In relation to our

work, model management operations such as model validation, comparison, merging and transformation are discussed further.

#### Model Validation

A fit-for-purpose model that captures a system's domain of interest needs to be well verified and validated. Model validation provides integrity to a software system under development using MDE. A model is "incomplete, contradictory and inconsistent when it leaves out information, shows difference in concepts and incomplete" respectively [16], [17]. In MDE, these outlined limitations on a fit-for-purpose model creates a system that does not capture the domain of interest of a system. By using model validation, any limitations can be detected, analysed and corrected using MDE approaches.

To have a validated and consistent model, constraints needs to be be specified on UML and MOF models using validation and constraint language such as the Object Constraint Language (OCL), an OMG standard tool. Pertaining to our work, a validation language highlighted in Subsection 2.1.2, called Epsilon Validation Language (EVL)is used and they help impose and evaluate constraints between models. By using EVL, dependencies can be supported between constraints thereby disintegrating complex constraints to simpler forms.

Other model management operations include merging, migration and comparison. Our work uses EMF compare for comparison of validated model to generate a Java OO code. EMF compare allows comparison of model where similar components in two models can be traced and analyzed [18].

### 2.1.5 MDE Approach and Tools

Effective approaches and tools are needed for MDE in other to aid guidance and means that will aid swift model engineering practices. This section discusses the level of abstraction involved in MDE, the method used to our advantage and the tools deployed to aid them.

#### Model Driven Architecture (MDA)

MDA is an approach to developing technological complex systems. It proffers a standard to models and modeling languages. This standards is reflected in representing and exchanging models (XMI), constraints specification (OCL), and transformation specification on models [19]. Model architecture allows models to be created at various level of abstractions mostly in standards formats like XMI. It is used for viewing, saving and exchanging software models and design. An architecture lay down guidelines and approaches for a MDE so as to enable the development of system from raw data and business logic, leaving behind the crucial implementation technologies. OMG development of model architecture allows the definition of

```
context OO!Class {

  // The name of a class should start with
  // an upper case letter
  critique NameShouldStartWithUpperCase {

    guard : self.satisfies("HasName")

    check : self.name.substring(0,1) =
      self.name.substring(0,1).toUpperCase()

    message : "The name of class " + self.name +
      " should start with an upper-case letter"

    fix {
      title : "Rename class " + self.name + " to " +
        self.name.firstToUpperCase()

      do {
        self.name = self.name.firstToUpperCase();}
    }
  }

  // A class must not directly or indirectly
  // inherit from itself
  constraint MustNotInheritItself {

    check : not self.inherits(self)

    message : "Class " + self.name + " inherits itself"

  }
```

Figure 2.5: EVL used to validate an OO model from Epsilon EVL

MDA standard where the use of a Platform Independent Model (PIM) and Platform Specific Models (PSM) is emphasized.

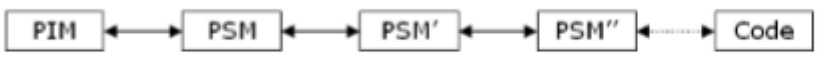


Figure 2.6: Interactions between a PIM and several PSMs

With PIM, abstract and agnostic-implementation of a system overview is provided while PSM provides more detailed system implementation. MDA allows transformations to be automated between PSM and PIM. The interoperability of MDE platforms can be developed with MDA.

### Standards for MDA and their Uses

OMG has proposed some set of standards to be used in MDA. These standards are the basis for defining metamodels and imposing constraints on models to make them conform to their metamodels. A pyramid is used to illustrate these standards representing different levels of abstractions in a model. Kleppe's *et al* [20] pyramid is shown below;

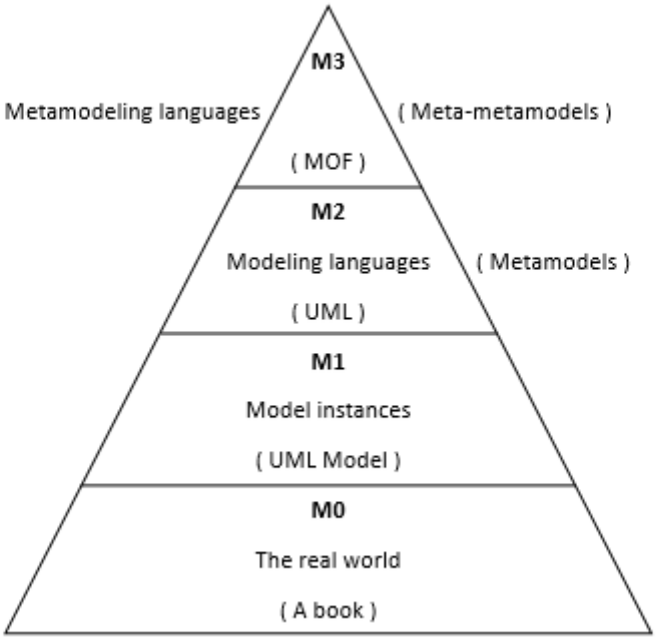


Figure 2.7: The four stages of model abstraction from Kleppe's *et al* [20]

The base of the pyramid, M0, depicts the domain of interest (real world). For example, a model of a software system representing the descriptions of the system's software representations is contained in M0. M1 is the representation of the M0 - a model of the M0 concepts. M1 is better shown as the representation of the model of the real world (contained in M0). M2 contains the modeling language (metamodels) used to define M1. M2 contains UML metamodels used in making M1 conforms to them. Lastly, M3 as an instance of M2 represents a metamodeling language (metametamodel) used in describing modelling language in M2. This pyramids represents the different level of abstraction observed in each domain of a system.

### Domain Specific Modelling

Domain-Specific Modeling (DSM) as a technique to a domain model provides a set of guidelines and process for developing systems using MDE. Here, approaches to transform domain to a code is provided. DSM lay emphasis on increased productivity by enabling software engineers to use models to specify solutions to specific domain. A domain-specific language is used to define DSM where in, domain experts develop solutions to a domain problem. Also, a code generator that allows the transformation of domain models to executable OO code is developed with DSM.

## 2.2 MDE Tools

MDE is developed by using a powerful tool that is susceptible to model interoperability. Software engineering process for MDE tools enable the development of tools that support interdependence between instances of a model and instances of their metamodels. This section reviewed the MDE tools used in our research.

### 2.2.1 Eclipse Modelling Framework

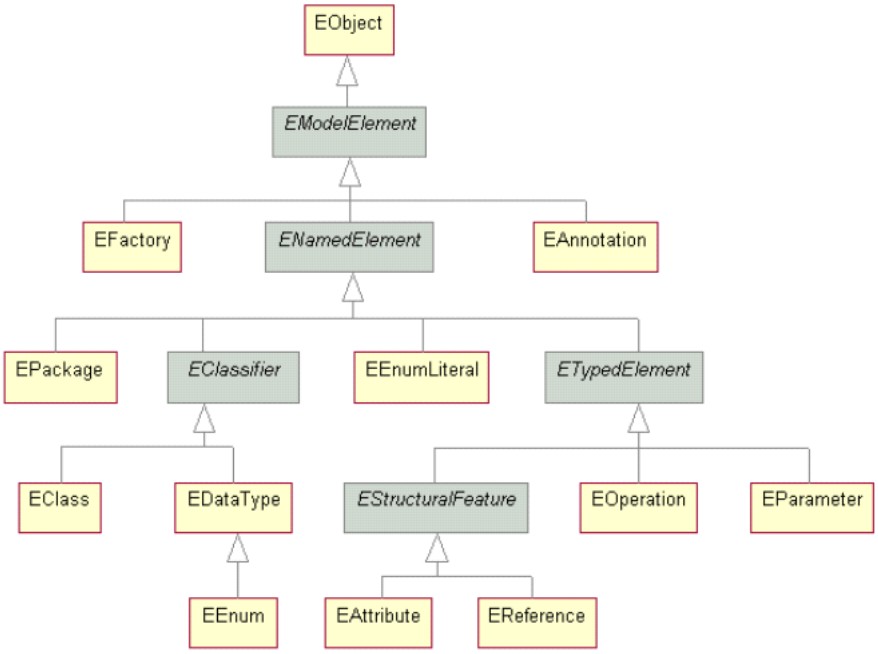


Figure 2.8: An Ecore Metamodeling Language taken from [21].

This is a framework that helps with the development and instantiation of metamodels. EMF provides support for MDE by facilitating code generation and providing a metamodeling language called Ecore. It is widely believed that EMF is the most widely used MDE modeling framework. EMF Ecore is used to define and instantiate the metamodel used in our work. EMF provides both a tree-based and

graphical metamodel editors which are subsequently used editors towards manipulating metamodels to provide needed and expected artefacts as an output.

In our work, with the definition of a metamodel, EMF enables the generation of a model editor that conforms to our metamodel. Likewise, a graphical modeling framework (GMF) can be created from metamodels instantiated with EMF. A model driven approach where several models are specified, merged and transformed so as to enable code generation is made possible from a graphical editor. Other associated tools used in aiding our work is discussed further.

### 2.2.2 Papyrus

Papyrus provides an environment where any kind of EMF model can be edited. It also supports UML and other related modeling language like SysML and Marte. Here, model (diagram) editor and support is provided for EMF and its tool integration. In Papyrus, UML metamodel and graphical models are used to define and modify any models thereby a behavior or structure model can be easily created. Users are able to define editors on UML 2 standard for DSL. In one of the approach used in our work, we deploy Papyrus to model our behavior diagram and their transformation to a structure diagram. Thus enables us to automate different UML models representing different domain of our system, model and metamodels.

### 2.2.3 Epsilon

The Extensible Platform for Specification of Integrated Languages for mOdel maNagement (Epsilon) [16] is a powerful tool suite used as a DSL for MDE. It incorporates many model management language and it is use for performing management tasks like transformation, validation and merging [Kolovos 2009].

Epsilon is deployed to manipulate models written in various languages thereby giving it an edge over other model management languages which are mostly constricted to some set of modeling technologies [22]. Models implemented using MOF, EMF or XML is widely supported by Epsilon.

The architecture of Epsilon allows the use of task-independent management tool. In our work, we use EOL to query our model, EVL for model validation and ETL for model transformation. The use of these tools allow us to deal with each model management operation in our work specifically.

## 2.3 MDE Benefits and Simulation Confidence

Among the benefits of MDE is the option of tool interoperability where tools via model can be interchanged. Using Ecore, EMF enables a reference implementation of MOF which serves as a foundation for various MDE tools.

With interoperability, model management operations can be performed on

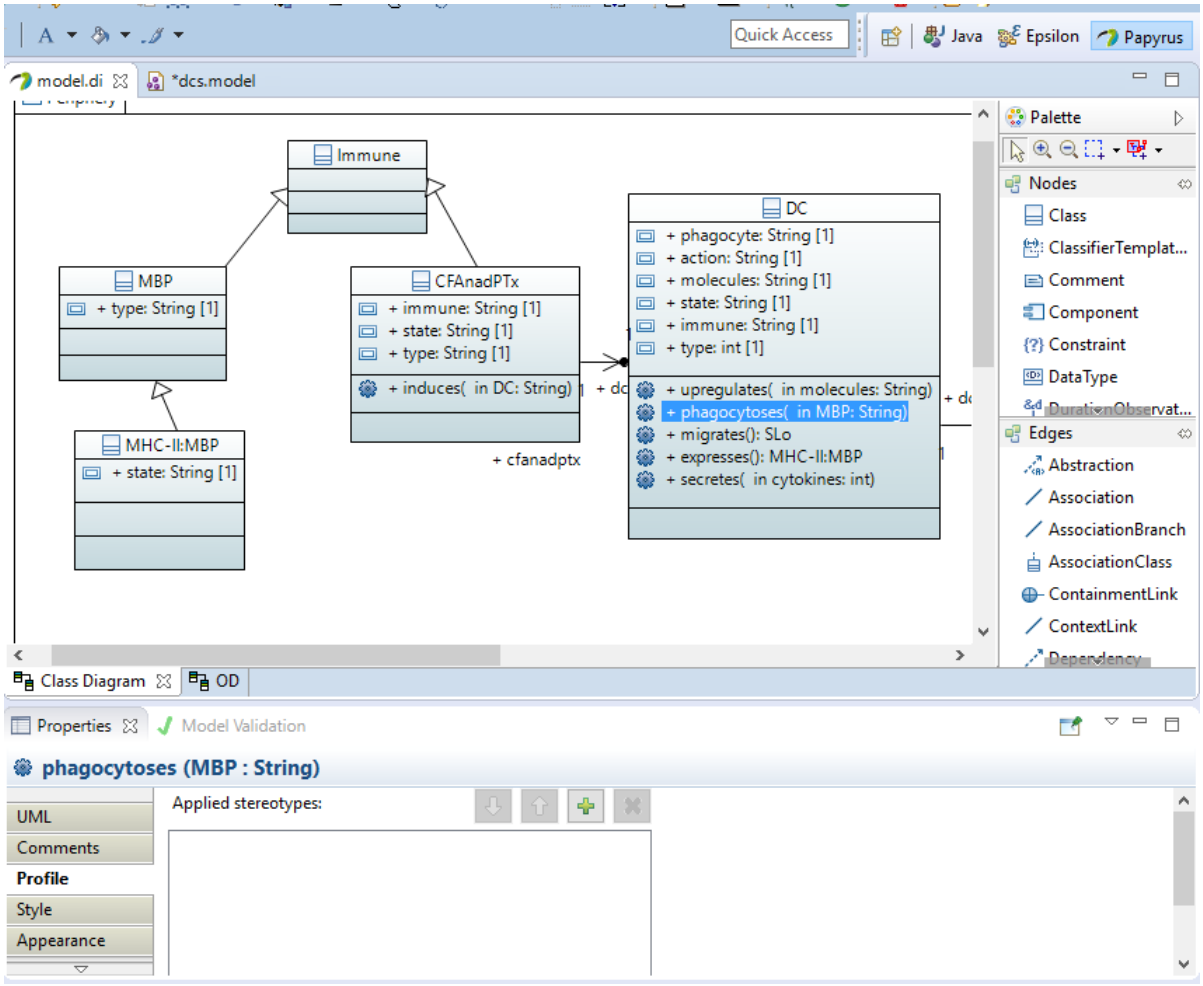


Figure 2.9: A Papyrus model environment.

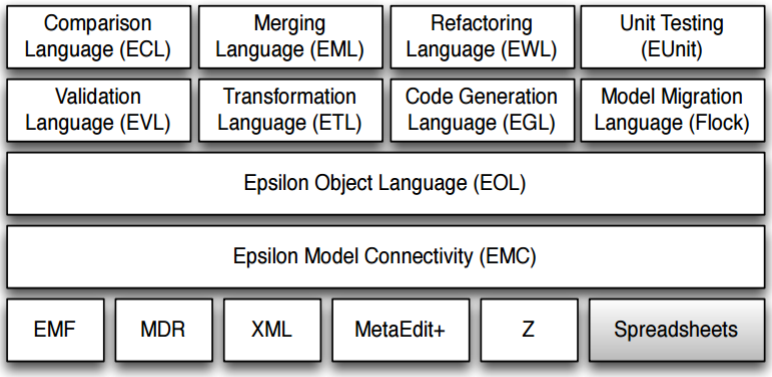


Figure 2.10: The architecture of Epsilon, taken from Eclipse Epsilon

wide selections of MDE tools. This operation is supported by powerful languages and tools like Epsilon.

As started earlier, MDE offers model management operations which enables

the automation of repetitive software engineering processes. This support for model management comes from different software migration and merging processes especially in software industry where development is not centralized.

Also, another benefit is the ability of MDE to initiate automation of error-free and complex elements of software engineering - a system scalability characteristics. For example, code generation can be used to produce repetitive code which cannot be altered to remove duplication (mostly for technological reasons) [11].

A major focus of model application in MDE, to an EAE immune system involve the use of diagrammatic approach to capture, connect and think around simulation. Models are noted with scientific and engineering assumptions thereby capturing the understanding of a system. However, models are sometimes generated and updated during simulation-centered research.

For confidence on simulation, there needs to be a conceptual format to the process where simulations are engineered and manipulated using MDE. Confidence is built when there is a modeling process that exposes how scientific facts are rendered into simulation who in turn helps mitigate unwarranted assumptions. A confidence in processes towards simulation helps strengthened structured models that can be validated and transformed  a pivot of our thesis. To have a confidence on models to be simulated, a conceptual framework by CoSMoS is reviewed.

## 2.4 YCIL: CosMoS Approach

Read's thesis [1] which we are exploring for our work uses modeling diagrams to but hand-craft code for simulation. The handcrafted code can be clearly written once and easily reused. However, handcrafted code might contain errors and may not reflect the actual representation of the models in contrast with a generated code. As MDE can both properly underpin the diagram notations and support transformation to code, we review the CoSMoS Approach being used for immune system simulation.

The CoSMoS project is an EPSRC funded project which aim is to develop generic tools and approach to aid modelling and simulation of complex systems [24]. The framework has various interconnected steps that help to guide our understanding of a systems complexity. CoSMoS framework can guide the UML, mathematical and *in silico* modeling for instantiating a system's domain on a high abstraction level [23].

Crucially, as part of the domain and platform modeling, a set of models that highlights the behaviors underpinning complexity of the system is created. These models are the basis for implementing the simulation and the result shows a pattern in the real-world system i.e. system of interest [25]. To achieve a fit-for-purpose simulation, the steps in the framework are explained.

**Domain** The real world system being modeled. This is the system of interest and information about it is provided by the domain expert.

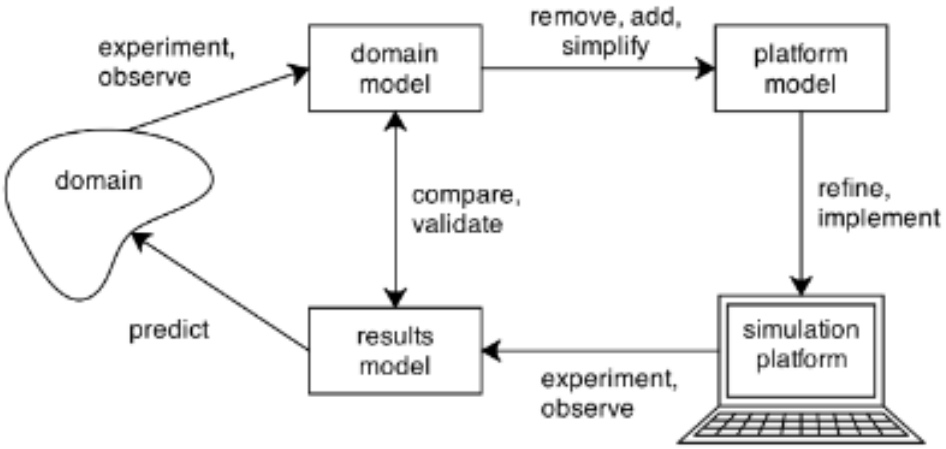


Figure 2.11: The complex system and Modeling Simulation(CoSMoS) framework followed in modeling our complex system from [23]

**Domain Model** This is a model that captures the scope of interest and their emergent properties. It identies and outlines the interactions and behavior of the domain[24]. It is based on the domain experts analysis. Systems to be simulated are model using a top-down methodology [26].

**Platform Model** This provides a design model that designs how the domain of interest will be simulated. A bottom-up approach is used here for easy construction of the different parts of the domain model to a platform model [26]. Simulation Platform By using software engineering processes, a simulator is developed from the platform model. The simulation platform provides executable implementation [24].

**Results Model** The result model is used as a comparison standard for the domain model so as to be able to interpret the simulation results in the context of the real domain [26].

# Chapter 3

# Domain Analysis and Observation

# Bibliography


[1] M. N. Read, "Statistical and modelling techniques to build confidence in the investigation of immunology through agent-based simulation," Ph.D. dissertation, University of York, Department of Computer Science, 09 2011.

[2] D. Moyo, "Investigating the dynamics of hepatic inflammation through simulation," Ph.D. dissertation, Department of Computer Science, York, 2014.

[3] W. Richard, "An agent-based model of the IL-1 stimulated nuclear factor-kappa b signaling pathway," Ph.D. dissertation, Department of Computer Science, York, 2014.

[4] J. Bézivin, "On the unification power of models," *Software & Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005. [Online]. Available: http://dx.doi.org/10.1007/s10270-005-0079-0

[5] J. Bézivin and O. Gerbé, "Towards a precise definition of the omg/mda framework," in *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, ser. ASE '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 273–. [Online]. Available: http://dl.acm.org/citation.cfm?id=872023.872565

[6] A. Rensink and J. Warmer, Eds., *Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4066. Springer, 2006.

[7] A. M. Starfield, K. Smith, and A. L. Bleloch, *How to Model It: Problem Solving for the Computer Age*. New York, NY, USA: McGraw-Hill, Inc., 1993.

[8] I. Kurtev, "Adaptability of model transformations," Ph.D. dissertation, University of Twente, Netherlands, 2004.

[9] R. F. Paige, N. Matragkas, and L. M. Rose, "Evolving models in model-driven engineering," *J. Syst. Softw.*, vol. 111, no. C, pp. 272–280, 01 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2015.08.047

[10] R. F. Paige, D. S. Kolovos, and F. A. Polack, "A tutorial on metamodelling for grammar researchers," Science of Computer Programming, to appear, Tech. Rep., 2014.

[11] L. M. Rose, "Structures and processes for managing model-metamodel co-evolution," Ph.D. dissertation, University of York, Department of Computer Science, 07 2011.

[12] S. Kelly and J. Tolvanen, *Domain-Specific Modeling - Enabling Full Code Generation.* Wiley, 2008. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470036664.html

[13] J. Alvarez, A. Evans, and P. Sammut, "Mml and the metamodel architecture," *Workshop on Transformation in UML, co-located with the European Joint Conferences on Theory and Practice of Software (ETAPS)*, 2001.

[14] D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing.* New York, NY, USA: John Wiley & Sons, Inc., 2002.

[15] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE Softw.*, vol. 20, no. 5, pp. 42–45, 09 2003. [Online]. Available: http://dx.doi.org/10.1109/MS.2003.1231150

[16] D. S. Kolovos, "An extensible platform for specication of integrated languages for model management," Ph.D. dissertation, University of York, Department of Computer Science, 2009.

[17] M. Elaasar and L. Briand, "An overview of uml consistency management," Carleton University, Tech. Rep., 08, sCE-04-18.

[18] C. Brun and A. Pierantonio, "Model dierences in the eclipse modelling framework," p. 2934, 2008, upgrade.

[19] I. Kurtev, "State of the art of qvt: a model transformation language standard," in *Applications of Graph Transformations with Industrial Relevance*, ser. Lecture Notes in Computer Science, A. Schürr, M. Nagl, and A. Zündorf, Eds., vol. 5088. Berlin: Springer Verlag, October 2008, pp. 377–393. [Online]. Available: http://doc.utwente.nl/62484/

[20] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

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

[22] D. Kolovos, R. Paige, and F. Polack, "The Epsilon Object Language (EOL)," in *Proc. European Conference on Model-Driven Architecture - Foundations and Applications (ECMDA-FA)*, ser. Lecture Notes in Computer Science, A. Rensink and J. Warmer, Eds., vol. 4066. Springer, 2006, pp. 128–142.

[23] Y. C. I. Lab, "Software engineering." [Online]. Available: https://www.york.ac.uk/computational-immunology/research/soft-eng/

[24] P. S. Andrews, S. Stepney, T. Hoverd, F. A. C. Polack, A. T. Sampson, and J. Timmis, "CoSMoS process, models, and metamodels," in *Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation*, S. Stepney, P. H. Welch, P. S. Andrews, and C. G. Ritson, Eds. Luniver Press, Aug. 2011, pp. 1–14. [Online]. Available: http://www.cosmos-research.org/docs/cosmos2011-proceedings.pdf

[25] P. S. Andrews, F. A. C. Polack, A. T. Sampson, S. Stepney, and J. Timmis, "The CoSMoS process, version 0.1: A process for the modelling and simulation of complex systems," Department of Computer Science, University of York, Tech. Rep. YCS-2010-453, Mar. 2010.

[26] P. Garnett, S. Stepney, F. Day, and O. Leyser, "Using the CoSMoS process to enhance an executable model of auxin transport canalisation," in *In Stepney et al*, pp. 9–32.