



Titre: Applying Model Driven Software Engineering Practices to
Title: Epidemiology

Auteur: Bruno Curzi-Laliberté
Author:

Date: 2024

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Curzi-Laliberté, B. (2024). Applying Model Driven Software Engineering Practices
Citation: to Epidemiology [Master's thesis, Polytechnique Montréal]. PolyPublie.
<https://publications.polymtl.ca/58343/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/58343/>
PolyPublie URL:

**Directeurs de
recherche:** Mohammad Hamdaqa, & Marios-Eleftherios Fokaefs
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Applying Model Driven Software Engineering Practices to Epidemiology

BRUNO CURZI-LALIBERTÉ

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie informatique

Mai 2024

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Applying Model Driven Software Engineering Practices to Epidemiology

présenté par **Bruno CURZI-LALIBERTÉ**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Bentley OAKES, président

Mohammad HAMDAQA, membre et directeur de recherche

Marios-Eleftherios FOKAEFS, membre et codirecteur de recherche

Michalis FAMELIS, membre

RÉSUMÉ

Ce mémoire motive, présente et évalue la conception et implémentation d'une plateforme de modélisation épidémiologique. Cette plateforme se base sur un métamodèle en suivant les pratiques d'ingénierie dirigée par les modèles. La plateforme s'utilise avec une interface graphique qui permet aux utilisateurs d'adéquatement représenter tous les aspects d'un modèle qui auparavant devaient être programmés. La plateforme supporte tous les modèles compartimentaux, épidémiologiques ou non.

L'objectif du projet est de s'attaquer aux problèmes de reproductibilité dénoncés récemment par plusieurs épidémiologistes. Certains modèles publiés ne présentent pas tous leurs paramètres ou équations, ou encore présentent des différences entre la présentation de leur modèle et le code source qu'ils fournissent. Notre plateforme rend le partage de modèle très simple, regroupant tous les artefacts nécessaires à l'exécution du modèle. De plus, les problèmes de différences entre représentation et implémentation sont inexistantes car l'implémentation correspond à la vue graphique utilisée lors de la modélisation.

L'objectif de recherche en ingénierie logicielle est une première application de l'ingénierie dirigée par les modèles dans le cadre de la modélisation épidémiologique. Dans le but d'atteindre une flexibilité maximale, la plateforme offre un métamodèle extensible qui, à l'aide de plugins, peut accepter de nouveaux types de compartiments et équations. Ces extensions peuvent être développées par des utilisateurs experts dans le but de les partager. Ces extensions permettent généralement de réduire la complexité des modèles où elles sont utilisées pour remplacer plusieurs éléments par un seul. L'extensibilité du métamodèle repose sur la composabilité des composants du métamodèle, soit les compartiments et équations.

L'évaluation de la plateforme aborde une étude de cas où un modèle de COVID-19 publié est reproduit dans notre plateforme. Le but premier est de démontrer que la modélisation de ce modèle est non seulement possible, mais simple, et génère les équations équivalentes à celles programmées par les auteurs du modèle étudié. Ces objectifs ont été vérifiés. Les critères de modélisation évalués sont la modularité et la modifiabilité. L'évaluation porte aussi sur les aspects fonctionnels du métamodèle tels que la complétude et s'il est approprié. Ces aspects sont subjectifs, mais l'évaluation suggère que non seulement le modèle de composabilité est adapté aux tâches de méta-modélisation envisagées, celui-ci permet de plus facilement bâtir les modèles et identifier plus tôt les fautes logiques grâce à l'approche graphique déclarative.

ABSTRACT

This thesis motivates, presents and evaluates the design and implementation of an epidemiological modeling platform. This platform is based on an extensible metamodel built using model driven engineering. The platform is used through a graphical interface which allows users to adequately represent all modeling aspects, previously requiring programming. The platform supports all compartmental models, epidemiological or otherwise.

The objective of the project is to tackle reproducibility issues recently denounced by many epidemiologists. Published models are sometimes presented without all relevant parameters and equations, or present differences between their presentation and the provided source code. Our platform renders model sharing trivial by grouping all required artifacts for model execution. Furthermore, differences between representation and implementation are completely avoided, as the graphical representation is directly used to implement the model.

The research objective for software engineering is a first application of model driven engineering to the field of epidemiological modeling. In order to maximize flexibility, the platform offers an extensible metamodel which, using plugins, can accept new types of compartments and equations. Extensions may be developed by expert users in order to share them. These extensions generally reduce model complexity where they appear, replacing multiple elements with a single one. Metamodel extensibility relies on composability of the components of the metamodel, the compartments and equations.

The evaluation of the platform proceeds with a case study where a published COVID-19 model is reproduced in our platform. The primary objective is to demonstrate that such modeling is not only possible, but simple, and generates equivalent equations to the ones programmed by the authors of the studied model. These objectives were verified. The modeling evaluation criteria were model modularity and modifiability. The evaluation also covers functional completeness and functional appropriateness of the metamodel. These aspects are subjective, but the evaluation suggests that not only the composability model holds for considered meta-modeling tasks, but it also allows to more easily build models and identify logic errors earlier with a declarative graphical approach.

TABLE OF CONTENTS

RÉSUMÉ	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF SYMBOLS AND ACRONYMS	x
CHAPTER 1 INTRODUCTION	1
1.1 Elements of the Problem	2
1.1.1 Reduce Reaction Time and Increase Productivity:	2
1.1.2 Facilitate Communication:	2
1.1.3 Reliability:	3
1.1.4 Reproducibility:	3
1.1.5 Extensibility:	3
1.1.6 Composability:	3
1.2 Research Objectives	4
1.3 Contributions	4
1.4 Outline	5
CHAPTER 2 BACKGROUND AND RELATED WORK	6
2.1 Basic Concepts	6
2.1.1 Epidemiology Concepts	6
2.1.2 Model Driven Engineering	8
2.2 Modern Compartmental Epidemiological Modeling	9
2.2.1 COVID-19 Nigeria Public Awareness Model	10
2.2.2 COVID-19 Mixing Models	10
2.3 Contact Mixing Patterns	11
2.4 Further Uses of Compartmental Modeling	12
2.4.1 Ecology	12
2.4.2 Climate Models	13

2.5	Epidemiological Modeling Reproducibility	14
2.6	Modern Model Driven Engineering	15
2.6.1	Ontological Approach	17
2.7	Composability	17
2.8	Related Work	18
2.8.1	Eclipse Spatio-Temporal Epidemiological Modeler	18
2.8.2	IDM-CMS	19
2.8.3	CSID tools	20
2.8.4	Kendrick	20
2.9	Key Takeaways	21
2.9.1	The Cost of Unique Features	21
2.9.2	Generic Platforms	21
2.9.3	Usability	21
2.9.4	Composability	22
CHAPTER 3	METAMODEL	23
3.1	Metamodel Requirements	23
3.1.1	Metamodel Extension Requirements	24
3.1.2	Metamodel Requirements Summary	28
3.2	Metamodel Implementation	29
3.2.1	EMF Wrappers	29
3.2.2	Metamodel Graphical Implementation	30
3.2.3	Metamodel Programming Implementation	33
3.3	Example Model	34
CHAPTER 4	PLATFORM	36
4.1	Platform Requirements	36
4.2	Platform Design	38
4.2.1	Project Creation	38
4.2.2	Metamodel Plugin Selection	39
4.2.3	Graphical Interface	40
4.2.4	Model Comparison	42
4.2.5	Parameterization	45
4.2.6	Simulation	46
CHAPTER 5	PLATFORM USAGE & EVALUATION	48
5.1	COVID-19 Case Study	48

5.1.1	Model Presentation	48
5.1.2	Platform Workflow	50
5.1.3	Modeling Using Our Platform	50
5.1.4	Model Parameterization	56
5.1.5	Model Validation	57
5.2	Evaluation Design	58
5.2.1	Domain Modeling Evaluation Practices	58
5.2.2	Software Quality Evaluation	60
5.3	Platform Correctness Evaluation	63
5.4	Platform Completeness Evaluation	67
5.5	Platform Appropriateness and Model Modularity and Modifiability Evaluation	67
5.5.1	Top Level Element	68
5.5.2	Comorbidity	69
5.5.3	Aging	70
5.5.4	Disease Model Link	70
5.5.5	Disease Model	71
5.5.6	Concluding Modularity	72
5.5.7	Concluding Modifiability	73
5.5.8	Concluding Appropriateness	73
5.6	Evaluation Results	73
5.6.1	Epidemiological Model Construction	74
5.6.2	Epidemiological Model Sharing	75
5.6.3	Epidemiological Model Reproducibility	75
CHAPTER 6	CONCLUSION	77
6.1	Summary of Works	77
6.2	Limitations	77
6.3	Future Research	78
REFERENCES		79

LIST OF TABLES

Table 4.1	Infectious Disease Modeling User Stories.	37
Table 5.1	Population Distribution	64
Table 5.2	Derivative of Susceptible Compartments	66

LIST OF FIGURES

Figure 3.1	Base Metamodel	30
Figure 3.2	Group Metamodel	31
Figure 3.3	Product Metamodel	32
Figure 3.4	Basic Flows Metamodel	33
Figure 3.5	Example Model	35
Figure 4.1	Normal Project Creation	38
Figure 4.2	Epidemiological Project Selection	39
Figure 4.3	Plugin Selection	40
Figure 5.1	Original authors' model representation	49
Figure 5.2	Model Structure	51
Figure 5.3	Model Structure and Flows	52
Figure 5.4	Source and Sink Selection	53
Figure 5.5	Stratified Model	53
Figure 5.6	Parameter Patterns Usage	56
Figure 5.7	Parameter Patterns Usage In Code	57

LIST OF SYMBOLS AND ACRONYMS

DSL	Domain Specific Language
EMF	Eclipse Modeling Framework
MDE	Model Driven Engineering
ODE	Ordinary Differential Equation

CHAPTER 1 INTRODUCTION

The COVID-19 pandemic is so far directly responsible for 7 million deaths¹, but could be attributed an even greater count². This pandemic has shown the need to quickly to answer important epidemiological modeling questions. Examples include the understanding of waves and variants, the impact of governmental interventions and estimating the peak number of hospitalization. These questions can be answered by making various epidemiological models which integrate our hypotheses.

Pandemic scale epidemics like COVID-19 are certainly rare, but epidemiological modeling can be applied to hundreds of diseases, which also are responsible for millions of deaths. For example, in 2021, tuberculosis was responsible for 1.6 million deaths worldwide³. Just like COVID-19, tuberculosis is a communicable disease, which means some understanding of both diseases' spread is transferable.

In 1900, tuberculosis was responsible for 194 deaths per 100,000 people in the United States⁴, this is similar to the mortality rates of COVID-19 in Canada. However, in 2021, just 0.2 deaths per 100,000 people died from tuberculosis in Canada⁵. The reason for this incredible drop in mortality is the Bacille Calmette-Guérin (BCG) vaccine, which has helped steadily reduce TB deaths since 1924, even though at the time, cases were still increasing⁶.

Our work can work in conjunction with the science behind vaccines. By using information we learned from tuberculosis, we can extrapolate models to COVID-19. It might sound simple as we expect vaccines to simply stop the spread of the disease, but vaccines are not perfect. For example, take the BCG vaccine, which only offers a 51% protection to tuberculosis⁶. This vaccine is similar to the COVID-19 vaccines which also offered partial protection.

By making various epidemiological models, we can develop best and worst case scenarios along with appropriate responses. These models may include different representations, vaccination, variants, age stratifications, hospitalization, etc. Developing and improving models takes time, effort and a great attention to detail, as models can become quite complex.

By applying Model Driven Engineering (MDE) to epidemiology with our platform, we offer a new way to create and manipulate epidemiological models. Our objective is to address

¹<https://COVID19.who.int/>

²World Health Organization - Estimating Excess Mortality

³<https://www.who.int/news-room/fact-sheets/detail/tuberculosis>

⁴<https://www.cdc.gov/mmwr/preview/mmwrhtml/mm4829a1.htm>

⁵<https://knoema.com/atlas/Canada/topics/Health/Risk-factors/Tuberculosis-death-rate>

⁶ Canada Public Health Services Immunization Guide BCG

problems faced by epidemiologists by facilitating model construction, model sharing and reproducibility [1–6]. This thesis presents a model driven compartmental epidemiological modeling platform prototype.

1.1 Elements of the Problem

This thesis operates in the intersection of software engineering and epidemiology. It presents the development of an MDE platform for epidemiological modeling and the development of an epidemic metamodel. The metamodel captures the main elements of compartmental epidemiological models and allow for better communication, reuse and encapsulation. Collaboration with an epidemiologist enabled us to identify epidemiology requirements and translate them to corresponding software engineering functional and non-functional requirements. The main focus of the thesis is a graphical environment for epidemiological modeling, but the platform also offers a separate parameterization and simulation environment.

1.1.1 Reduce Reaction Time and Increase Productivity:

During an epidemic, as we saw during COVID-19, it is important for governments and populations to react quickly and decidedly to forecasts to avoid crisis. Epidemiologists model various possible scenarios of the spread of the disease that account for various possible government decisions that affect the rate of spread.

This can be a lot of work, for teams or individuals. Our platform can help accelerate various aspects of the work done by epidemiologist. By improving modeling and simulation we reduce the overall response time to integrate new data or hypotheses into models.

1.1.2 Facilitate Communication:

Study of a disease, of its propagation mechanics and parameters of a given population requires considerable time and often can't be shared for direct use by other epidemiologists. To accommodate for model sharing, it is necessary to standardize representations and use cases of disease and population data.

In addition to model and data sharing, it is expected to be able to share products of the model, such as curves, diagrams, evaluations, comparisons, etc, in a way that connects them back to the original model for reproducibility and traceability. These views of a model can be useful for many individuals involved in the decision process without necessarily being epidemiologists. These views are to be produced automatically to minimize risk of error or

misleading information.

1.1.3 Reliability:

Since epidemiological modeling is not an individual task, models can change quickly and are at risk of being broken. The use of an MDE framework such as Eclipse Modeling Framework (EMF) allows us to validate structural consistency of models. More validation mechanisms are implemented, notably model comparison, to allow modelers to detect errors.

1.1.4 Reproducibility:

It is necessary for the process of sharing models or simulations to be simple in order to quickly compare or verify results. This is made even more important when accounting for version management which requires frequent verification in order to avoid breaking changes when modifying models or simulation parameters.

1.1.5 Extensibility:

It is unwise to attempt to create a tool with a limited set of features in hopes that every epidemiological model can be represented using it. We instead provide an extensible metamodel that is easily extensible for simple use cases.

Extension of the metamodel allows user-developers to create new types of equations or compartment structures. Using this functionality, it is possible for a user-developer to enhance the metamodel to reduce the complexity at the model level. Typically, a metamodel extension is expected to replace a configuration of many elements from the base metamodel with a single element.

The problem lies in identifying the best way to define operations and types in order to make our metamodel extensible. This requirement is tied to composability, as both of these concerns affect the requirements of the base type system.

1.1.6 Composability:

Composability is a fundamental principle in software engineering that refers to the ability of components or operations to be combined to create complex systems or structures. It allows building blocks to be arranged in different configurations to create larger, more sophisticated blocks without fundamentally changing the individual components themselves.

Supporting composable models is a key part of our platform philosophy. In principle, if applied correctly, composability offers for the user the ability to abstract the fact that the model is larger than the part he is working on. This offers challenges in software design at the metamodel level to accomodate for an adequate composability model.

1.2 Research Objectives

By applying MDE to epidemiological modeling, we aim to improve:

1. Epidemiological model construction
2. Epidemiological model sharing
3. Epidemiological model reproducibility

1.3 Contributions

My contributions to the project in terms of research and application are as follows.

- A base metamodel closed for modification but open for extension and basic metamodel extensions for fundamental operations, as well as guidelines for users to develop their own extensions.
- A graphical epidemiological modeling environment in which all structural and behavioural epidemiological concerns are represented, eliminating programming tasks during modeling.
- A simulation and parameterization environment in Python, favorable to data-science-like notebooks, which accepts the generated simulation artifacts from our platform.
- An evaluation of the solution and ideas proposed in this thesis through a comprehensive case-study consisting in implementing a published COVID-19 model.
- An open source GitHub repository⁷ and a YouTube channel⁸ to distribute information.

⁷<https://github.com/yorku-ease/Model-Driven-Epidemiology>

⁸<https://www.youtube.com/@Epi-model>

1.4 Outline

The second chapter, background and related work, identifies relevant work in epidemiology, Model Driven Software Engineering and compartmental modeling. Relevant includes scientific research, but also projects similar to our platform. The third chapter, Metamodel, presents the set of concerns related to metamodeling and modeling, as well as our approach and solution. The fourth chapter, Platform, presents requirements and design of components not directly related to metamodeling or modeling. It covers subjects such as epidemiology in terms of user stories, model comparison, simulation and parameterization. The fifth section, Platform Presentation & Evaluation, presents the usage of our platform to create epidemiological models through a thorough case study re-implementing a COVID-19 model used by Ontario. Finally, the conclusion summarizes our work, limitations of the proposed solution and future research perspectives.

CHAPTER 2 BACKGROUND AND RELATED WORK

2.1 Basic Concepts

The following is an overview of concepts relevant to epidemiology or software engineering, preliminary to the understanding of the thesis.

2.1.1 Epidemiology Concepts

Epidemiological modeling is based on simple concepts which, once used in conjunction, allow for accurate modeling of the evolution of a disease in a population over time. We present the fundamental tools used by epidemiologists to build their models. These concepts are directly embedded into our platform in terms of features, design and architecture.

Compartmental Models

A simple epidemiological model may classify each member of a population along four groups: Susceptible, Exposed, Infectious, Recovered. This is known as the SEIR model [7]. The objective is to model the distribution of the population along the compartments over time. Models like these are known as compartmental models, they model the population as a whole instead of individuals [7]. This allows simulations to use the quantities of individuals as a real number instead of a natural number, for a large population, and treat the problem as continuous and differentiable.

Labelling individuals is not unique to compartmental modeling. In agent based modeling, individuals are labeled and their label may change [8]. In compartmental modeling, individuals are placed in compartments, not to be treated individually. Models are built using labelled compartments connected by flows, which direct the population values from one compartment to another following various equations.

Stratification

The basic form of the SEIR model is not sufficient for accurate models, but it can be improved easily. A common way to do so is known as stratification: stratifying the model along different axis that have an interaction with the disease spread. Some examples are: comorbidity, age, geographical position, remote work, etc. The use of these criteria is to create more precise compartments to make use of all available data and tune individual parameters

more accurately. The resulting compartmental model labels are expanded using a Cartesian product. An SEIR model stratified with three age groups and three geographical positions yields a model composed of $4 * 3 * 3 = 36$ compartments. One of the 36 compartments could be [*Susceptible, Adult, Montreal*].

Stratification with equations is similar to a Cartesian product of graphs [9], but equations add complexity to graph edges. It is also possible to stratify a compartment instead of the entire model. For example: stratifying the infectious state I of SEIR into {Symptomatic, Asymptomatic} or {Hospitalized, Not Hospitalized}. There can be disadvantages to doing so. For example, it might have been interesting to represent hospitalization for the entire population, not just the infectious population. Still, a non fully stratified model may be appropriate, as the resulting model is more easily understood and requires less computation.

Ordinary Differential Equations

To represent the evolution of different characteristics of the population, such as disease infection or simply aging, most models use Ordinary Differential Equation (ODE)s [10]. These equations can be thought of as flows, pointing from a compartment to another. Simple models make use of three types of flows: Rates, Batches and Contacts.

- Rate: Used to represent an evolution of a state, for example aging or becoming infectious once exposed. The amplitude of the flow is directly proportional to the source compartment. Transitions from a state to another require symmetrical operations, in this case a negative derivative for the Exposed compartment and a positive for the Infectious:

$$\begin{aligned}\frac{d}{dt}E(t) &= -E(t) \cdot c \\ \frac{d}{dt}I(t) &= E(t) \cdot c\end{aligned}$$

It is a simple representation for this specific case, but as noted earlier, simple concepts can be composed into more complex ones. For example, simply using identical rates in sequence will produce a Poisson distribution which represents much more adequately the reality. The simple version has a non negligible chance of a person becoming exposed and becoming infectious on the first day. A Poisson distribution can be much closer to what usually happens to exposed individuals, for example having a 95% chance of becoming infected between three to five days and a negligible chance earlier than three days.

- **Batch:** Used to represent an event happening over a period, for example when a variant is discovered, we can represent the apparition of this variant during a single day, infecting 1000 people. This means over that period we have a constant derivative:

$$\frac{d}{dt}S(t) = \begin{cases} -1000, & \text{if November 1st 2021} \\ 0, & \text{otherwise} \end{cases}$$

$$\frac{d}{dt}I(t) = \begin{cases} 1000, & \text{if November 1st 2021} \\ 0, & \text{otherwise} \end{cases}$$

- **Contact:** Used in essentially every epidemic compartmental model, the contact's primary role is to represent exposure to a disease. This flow approximates physical interactions between susceptible people and infectious people, exposing some of the susceptible population to the disease. This contact is done by multiplying the populations of the susceptible and infectious compartments [11,12]. We describe this flow as such:

$$\begin{aligned} \frac{d}{dt}S(t) &= \frac{-I(t) \cdot S(t) \cdot c}{N(t)} \\ \frac{d}{dt}E(t) &= \frac{I(t) \cdot S(t) \cdot c}{N(t)} \end{aligned}$$

Models may represent thousands of infectious people and millions of susceptible for a given scenario and vastly different values for a different scenario. A division by the total population at time t , $N(t)$, accounts for this disparity without changing the value of the constant c . c can then represent infectiousness based solely on population density and disease properties, instead of also embedding information about the specific population model.

2.1.2 Model Driven Engineering

MDE is a software development approach which aims to facilitate the development process by making use of an ensemble of models during development. These models are used for communication, code generation and at runtime. Code generation and use at runtime is possible by making use of frameworks which guide development, such as the Eclipse MDE Framework: EMF [13]. For our purposes, use of EMF is straight forward:

1. Creation of metamodels that represent the domain using types and relationships between types;

2. Customization of the EMF generated code by adding method signatures to interfaces and implementations to concrete types;
3. Creation of one or more software tool for model creation, modification or usage, which make use of the generated and written code.

Each step requires care, since subsequent steps rely on it. There is then a composed additional cost to modify the metamodel once user models or code have been created. Typically, to ensure that errors are not propagated through the entire workflow, domain experts are required to present concerns with the metamodel. In the case of EMF, the metamodel is a UML diagram, it is easily understood and contains visually all of the information about the fundamental requirements and assumptions of the software. Mistakes can be caught early on by domain experts as opposed to classical approaches where assumptions can be hard to identify in either documentation or implementation.

Resulting MDE software developed using metamodels has advantages. It is inherently safe and simple to incorporate corresponding models into the software since they conform to the metamodel. This makes for a reliable method of storing interoperable artifacts and fosters model sharing. This is especially interesting in our context as we develop an open-source platform and want to accommodate for adoption and extension.

2.2 Modern Compartmental Epidemiological Modeling

To better understand the models we need to support, we identified existing COVID-19 models, since they are some of the most recent. For this task, we used models published in the sixth volume of the *Infectious Disease Modelling Journal*¹, published in 2021.

The identified models serve only as examples, through which we can illustrate the general modeling process. Compartmental modeling for epidemiology involves a minimum of three steps. First, to construct a given model, researchers identify a structure, usually a variant of the SEIR model, which incorporates some assumptions about the population or the disease. This usually includes stratifying susceptible, exposed or infectious compartments into more specific labels, such as symptomatic, hospitalized, etc. Then, compartments are linked by flows. The configuration of the flows embeds some knowledge about the population or the disease. For example, models may allow a flow from asymptomatic to symptomatic but not the other way around. These decisions require field studies in hopes of closely matching the disease mechanics. Finally, each flow is parameterized, either according to known data or by using optimization techniques.

¹<https://www.sciencedirect.com/journal/infectious-disease-modelling>

2.2.1 COVID-19 Nigeria Public Awareness Model

A well cited model from Musa, Qureshi, Zhao, Yusuf, Mustapha and He presents multiple forecasts and recommendations based on studies of the effect of public awareness on the basic reproduction number, R_0 , in Nigeria [14]. The basic reproduction number, R_0 , is commonly used to determine how fast a disease is spreading at the beginning of a pandemic. $R(t)$, the effective reproduction number, evolves over time. Both refer to the number of subsequent infections caused by one newly infected person.

The proposed model for COVID-19 in Nigeria is a variation of the SEIR model. They separate Susceptible into two: Those who follow preventive health measures, such as quarantine and usage of face masks, S_a , and those who don't, S_u . They separate Infectious into four: Asymptomatic, A , Symptomatic, I , Hospitalized for mild symptoms, H_m , and Hospitalized for severe symptoms, H_s . They also make use of a Death compartment (only COVID deaths), D , for a total of 9 compartments: S_a , S_u , E , A , I , H_m , H_s , R & D .

The model uses 15 equations to describe the flows between compartments. The authors use many variables to describe influence of disease factors or preventive health measures and note the impact of each variable on R_0 .

Modeling social behaviours such as quarantine at the same level as the disease states such as susceptible or infectious is a widespread practice. Experimenting with different kinds of models and approaches to representing concerns such as these variations to the SEIR model helps define requirements and use cases for our modeling tool.

2.2.2 COVID-19 Mixing Models

A publication from Gumel, Iboi, Ngonghala and Elbasha presents three COVID-19 models. First, a basic compartmental model, assuming a homogeneously-mixed population. Second, a model with heterogeneously-mixed populations. Third, a model using a theoretical vaccine, as, at the time, no vaccine was available. The authors introduce parameters for the effectiveness of lockdown and the compliance of face mask usage to project the impact on the basic reproduction number, R_0 , and on total deaths from COVID-19. This is interesting for our purposes as supporting model evolution is part of the scope of the research. These models are not presented as evolutions but are related nonetheless.

Their basic compartmental model is composed of the following compartments: susceptible, S , exposed, E , symptomatically-infectious, I_s , asymptomatically-infectious, I_a , hospitalized or self isolating, I_h , recovered, R , and dead from COVID-19, D . For a total of 7 compartments: S , E , I_s , I_a , I_h , R & D . The transmission dynamics of their model makes use of 9 flows.

It is quite similar to the Nigeria model, but makes different assumptions about which behaviors impact the model, though it seems objectively simpler. For example there is a single Susceptible compartment instead of two and three Infectious compartments instead of four. The idea of grouping hospitalized and isolating people is not unique to this model. Grouping otherwise unrelated elements because they share behaviour may make models harder to maintain, but may allow for faster development.

Their second model makes use of stratification to use different rates of infection, progression and recovery. The stratifications are theoretical, but the authors note that age and social, cultural, demographic, or geographic factors influence contact patterns and disease periods. Once stratified, the same equations apply to the model, except for their varying parametrization. For example, stratifying N times yields N times more recovery rates. The exception is the exposure contact, since we now have N times more infectious and N times more susceptible compartments, we end up with N^2 times more contacts between the compartments. Each of these contact is parameterized with some factor of how likely these groups are to interact and how likely interactions produce exposure to the disease.

The interesting thing about this approach is that these contact behaviours can be measured or inferred based on studies or other models. Having the data from COVID-19 for example might give us an edge over the next epidemic to parameterize models more accurately. In theory, it is also possible to re parameterize the model for a different geographical region simply by modifying the contact pattern parameters.

Their final model includes vaccination. Surprisingly, they only add a single compartment, separating Susceptible, S , into unvaccinated and vaccinated, S_u & S_v . In hindsight we know that vaccination has an impact on how we are affected once infected with COVID-19, notably greatly lowering the mortality rate. This means there could have been variations on the Infectious compartments as well, but it was a demonstration of a theoretical vaccine. This is another simple example of oversimplification which may cause more work later.

2.3 Contact Mixing Patterns

Contact patterns describe how we parameterize contacts between susceptible and infectious people when using stratifications. As a reminder, a stratification is a criterion used to label our data more accurately to make models behave more accurately. These can include sex, age, geographical position, etc. There can be various ways to parameterize contacts between different sex or age groups. [15] identifies normal mixing, reduced mixing, proportional mixing, and segregate mixing. Essentially, normal and reduced mixing make use of data from

In practice, we can use the same equation with a different valued multiplier to implement each pattern. It provides us with a use case for developing a family of contact equations and defining how they operate if we wish to. Our design allows implementing a specific pattern by using the adequate equation templates.

2.4 Further Uses of Compartmental Modeling

We have designed our platform to be able to build any kind of compartmental epidemiological model. However, as a result, we are able to produce various non-epidemiological models. Ecological or social dynamic models are similar to epidemiological models, so we can expect some developments on this end. Ideally, our platform would support any kind of compartmental modeling, which indeed applies to many domains outside epidemiology. Exploring different applications of compartmental modeling will help make the platform as generic as possible. It should also be noted that ecological models are required in the modeling of many diseases, such as vector-borne diseases. These models typically also integrate climate so we should consider the implications on our platform to accommodate for climate model integration.

Social Dynamics

Social dynamics is a field in which compartmental modeling can be applied in a near identical way as epidemiology [16–18]. Social dynamics are easily understood as compartmental models, since simply considering an opinion as a disease closely resembles an epidemic. Behavior, knowledge and opinions can be represented by social dynamics. Simple examples include opinion spreading between friends while complex examples might evaluate the impact of spending time on social media on the use of tanning salons.

2.4.1 Ecology

Ecological models can be any kind of ecosystem, such as predator-prey models [19] or climate dependent population models. Ecological models are even used in epidemiological modeling as they are required to accurately model vector-borne diseases. Vector-borne diseases refer to diseases that are carried by arthropods and transmissible to humans. To model these diseases, epidemiologists or ecologists need to model the arthropod populations, for example a species of mosquitoes that are known to infect humans with malaria. Surprisingly, mosquito population models are more complex than human population models. [20] presents a flowchart for modeling *Anopheles* populations which is similar to a climate dependent compartmental

model. This model requires certain operations over time in the population, such as migration, laying eggs, growth stages, etc. These can all be represented using flows in our platform, such as rates or by making use of extensions. This thesis focuses on simpler behaviours than this insect model, but it seems likely that the composability model adequately supports complex flows required for such modeling tasks. By using a generic compartmental modeling approach, we can adequately model mosquito populations using software originally built for modeling infectious diseases.

Our models make use of ordinary differential equations, treating population values as continuous, so small populations might be at risk of being inappropriately modeled. This means that not every model that can be represented in our platform should be. For example, it is unlikely to be a good idea to model snow leopards using compartmental model, as there are only 6500 remaining snow leopards, occupying 1.8 million square kilometers². When modeling small populations, agent based model and knowledge about specific individuals can affect the result in significant ways, which are abstracted when working with large populations and continuous values. Agent based models and compartmental model present only a few differences, not from a mathematical perspective but from a modeling perspective, as the structural aspect is unchanged but the behavioural is different. It might be relevant to support both in the future to better accommodate for highly specialized models. In theory, if such extensions are made to the platform, hybrid models might be achievable.

2.4.2 Climate Models

Ecological and epidemiological models that include climate, such as models parameterized with humidity, rain, wind and temperature, are especially relevant in the context of climate change [21–23]. We do not expect to model climate in our platform, rather we expect users to integrate existing climate models as data into the simulations by merging geographical climate models with geographical epidemiological models. Platform users can merge the epidemiological parameters and the climate parameters using the generated simulation artifacts in the simulation notebook. Just like typical epidemiological models require parameters such as contagiousness, these climate aware models will include parameters like humidity and temperature. One main difference between a contagiousness parameter and temperature is that the temperature is dynamic while we expect contagiousness to be static for simplicity.

²<https://wwf.ca/species/snow-leopards/>

2.5 Epidemiological Modeling Reproducibility

In order to understand the scope of the reproducibility issues, first presented to us by the expert epidemiologist, he also provided a list of contributions to the definition of the problem by his peers. Most of these contributions are not in the form of articles. Many experts in the field criticise the lack of reproducibility or reliability of published articles. Some studies were made to quantify this along various different evaluation criteria, such as this study of 29 published models using 27 criteria [1]: "For example, seven (24%) of 29 models did not report the equations used, nine (31%) did not report their estimated parameters, 13 (45%) did not share all of their longitudinal data, and 15 (52%) did not report their code". This study received mixed feedback [2], explaining that the evaluation should follow MINIMAR (MINimum Information for Medical AI Reporting) standards [24]. Our platform can directly help for the third and fourth requirement aspects suggested by MINIMAR:

- (3) provide detailed information about the model architecture and development so as to interpret the intent of the model and compare it to similar models and permit replication.
- (4) transparently report model evaluation, optimization, and validation to clarify how local model optimization can be achieved and enable replication and resource sharing.

The third point is directly addressed by the platform, since we represent models graphically using easily understood building blocks. The fourth is indirectly addressed, as our platform outputs simulation artifacts, at least as of now, given it is in prototype state. Similarly to machine learning or other data science fields, researchers will have to write code to manipulate and integrate data to models, so part of the responsibility is to the researcher. One of the future objectives of the project is to offer a Python library or other means of easily tuning models. Using a library implies writing less user code, making it more easily understood and less error prone.

A recent survey of the field concludes that more can be done on the front of connecting a model's structural qualities to its behavior to develop readers' intuitions [25]. A platform with a graphical interface to visualize model structure and use comparison functionalities enables explaining how these features distinguish it from other models and reason about its behavior. The U.S. Government Accountability Office suggests: "The Secretary of Health and Human Services should direct CDC to establish guidelines that ensure full reproducibility of CDC's research by sharing with the public all permissible and appropriate information needed to reproduce research results, including, but not limited to, model code" [5]. The

need is clearly identified for reproducibility, but we also hope to improve productivity and reduce the amount of user code.

2.6 Modern Model Driven Engineering

A search for MDE tools for epidemiological modeling, concluded that this was the first application of MDE to the field of epidemiological modeling. This is a concern, since it makes it hard to compare our work to prior work. The review was extended to include MDE in general, in search of guidelines for applying MDE to new domains. A field survey from Silva presents MDE's identity [26]. First, the idea that a model is a representation, or rather an approximation, of a system under study. In our case we are trying to approximate the underlying mechanics of an epidemic by characterising the population with a compartmental model. Second, the use of the term "language" to designate any representation of a model, be it in a textual Domain Specific Language (DSL) or a graphical interface.

The author explains that there is a relationship between formalism and execution of the model, in our case we create non executable models. Our models do have a runtime operation, which is to generate the simulation artifacts, however, this operation is stateless. Non executable models are essentially data, inherently simpler than executable models. It simple to guarantee formalism as models can be built using a purely declarative approach. We use a concrete syntax that closely models the abstract syntax.

The author identifies available tools for various MDE tasks, including EMF, and some solutions made available as Eclipse plugins. The use of Eclipse plugins, either with or without dependency to EMF, as the deployment mode for MDE tools is a recurrent theme, [26–29], though some practitioners are concerned that not enough tools provide web alternatives [27]. EMF is a tried and tested framework which seemed appropriate for our simple modeling task. Usage of EMF pairs well with Sirius ³, a solution for our needs of a graphical interface. Using EMF also points to using an Eclipse Plugin to distribute our work. Silva's field review supports these technologies and they are the most easily accessible. Furthermore, the review mentions Obeo Designer ⁴, the distribution of Eclipse which was used during development of our platform, simplifying the integration of EMF and Sirius.

Among other relevant mentioned technologies are acceleio ⁵, or the Acceleio Query Language, AQL, used to query elements in the diagram by Sirius, which offers an AQL scripting interface,

³<https://www.eclipse.org/sirius/overview.html>

⁴<https://www.obeodesigner.com/en/>

⁵<https://www.eclipse.org/acceleio/documentation/>

as well as OCL ⁶ which was considered but not used in the current implementation. The author notes that these tools gain from "open and strong community support", which was indeed of great help during the project, through the use of forums.

One of the features of the project is a DSL. The implementation does not include a textual DSL, but rather a graphical DSL, since it was deemed more useful. A recent survey of existing and currently maintained DSLs, [30], gives some insight about the evolution of our graphical DSL. It presents two points our project can gain from considering.

The first point is identified by one of the survey respondents, explaining that one of the reasons for a breaking change in their DSL was to accommodate a change in research focus to a new technology. In our case, breaking changes might be required if we make the decision to output tensor operations instead of individual equations from our model.

Tensor operations are very practical for epidemiological modeling, as they allow for faster computations. Consider rates of transfer from one group of compartment to another, in tensor form this is an element-wise multiplication of a population vector with a vector of coefficients, followed with a vector subtraction and addition. In non tensor form, this operation is a for loop which applies this multiplication, subtraction, addition individually. This is especially problematic as the simulation environment is in python, which is inefficient, but in any language is definitely slower than relying on the GPU for a tensor operation.

As of now, the modeling environment is adequate for outputting tensor operations, however, the artifacts that are generated cannot be translated into tensor form. It is unclear how well the artifact composability model would hold under this change. From a computing perspective, not using tensor operations is simpler, but a waste of time and energy. Generating correct equations rather than efficient equations is the first concern, so we have not implemented tensor operations yet.

The second relevant point brought up by the authors is not specific to DSLs. The authors suggest accepting the absorption of their work by another project. This could be viewed as a defeat but also as a victory. It was already suggested that if a similar project was found, our work could be to add functionalities to it instead of starting from scratch. No projects similar enough to our vision were found, so the idea has been put aside. In the event where our work coincides with other platforms, since our workflow is decomposed into modeling, parametrization and simulation, it should be possible to implement model conversion to and from other platforms' formats at any given step.

⁶<https://projects.eclipse.org/projects/modeling.mdt.oc1>

2.6.1 Ontological Approach

Ontological approaches are used in model driven engineering in order to translate the domain into a discernible structure. The alternative to this approach is to approximate the domain based on knowledge from individuals, which is unproductive [31].

For our sake, an ontological approach was considered, but ultimately abandoned. Examining different epidemiological models did not allow us to connect what the researchers are modeling and what the epidemiologist in our group was hoping to build. This is a fundamental disconnect from an ontological approach, in which sets of systems are used to draw an ontology from their components. This idea will be presented during the evaluation section of our platform, as we attempt to determine the appropriateness of the novel tools we have developed. These tools are an extension to the current epidemiological modeling practice, hence, they could not have been synthesized from current models.

2.7 Composability

As mentioned previously, one of the software engineering aspects of the project is composability. We aim to produce easily extensible software, enabling plugins to define new components. These components can be composability tools or regular components. The aim of composability is for every composability tool to integrate any other composability tool or regular component.

Some forms of composability are simple, by implementing a single composite type. For example: for a set of *Command* classes with an *execute* method, one could create a *CommandGroup* class which has a list of commands and calls *execute* on each of them. Our composability model is more complicated since the return type are simulation artifacts for the epidemiological model. In a presentation at CppCon2021 ⁷, Ben Deane presents his views on composability in C++ ⁸ [32]. According to Deane:

- Composability relies on Uniformity [32, p.90], which is achieved when
 - functions take the same types
 - functions return the same types
 - functions take the same types as they return (or related ones)

[32, p.97];

⁷<https://cppcon.org/program2021/>

⁸<https://www.youtube.com/watch?v=zclxRbh4AN0>

- composable interfaces should support a "Zero" type [32, p.56-72];
- composable interfaces should support a "Pass-through" or "Fallback" type [32, p.73-86];
- In classical Object Oriented Programming, adding types to hierarchies is simpler than adding operations for each type. This can be solved using the visitor pattern but is a trade-off where it is now harder to add types. [32, p.109-112];

This presentation enables a very fundamental description of the problem. Our composability model is similar to what is described by Deane, but instead of functions, we compose objects. Our objects provide compartments and equations. For any composability tool to provide its compartments, it has to gather the compartments from its children, the same is required for its equations.

2.8 Related Work

This section explores existing solutions for epidemiological modeling.

2.8.1 Eclipse Spatio-Temporal Epidemiological Modeler

Spatio-Temporal Epidemiological Modeler ⁹, STEM [33], is part of the Eclipse ecosystem. It is the only epidemiology tool based on Eclipse. STEM is interesting in the sense that it is much closer to what people imagine when thinking of an epidemiological modeling tool than what we are building. As the name suggests, it is used with geographical data and already offers many scenarios of disease spread in various countries. The lead developer published a tutorial video ¹⁰. In it, you can see the spread of a disease through China, for example at 16:30. It is complicated to use and crashes when doing operations in the wrong order. It integrates many features in parallel such as the model, triggers, scenarios, parametrization, simulation, etc. This makes it complicated to try a simple use case. Creating a model from scratch is an expensive task and it is easier to adapt existing models, but they don't necessarily fit the need. It is potentially better for specifically geographical models, but for our needs this is not a concern, so we did not look into implementation details. The large community and user models accompanying STEM is definitely the strongest argument for it. For example, STEM offers the Global Air Travel Network, see figure 3 in [33], calibrated with passenger travel through commercial airports.

⁹<https://projects.eclipse.org/projects/technology.stem>

¹⁰<https://www.youtube.com/watch?v=2bm-1ktbsmU>

STEM is specifically engineered for geographical models and modeling the spread of diseases over large areas. While there is definitely value to these features, this is not the direction that this project is moving towards. Essentially, the features proposed by STEM are not optional and draw away from the simplicity we aim to achieve. The platform we propose is strictly for compartmental modeling at a fundamental level, abstracting geographical aspects.

2.8.2 IDM-CMS

The Institute for Disease Modeling Compartmental Modeling Software, IDM-CMS, allows users to run simple compartmental models. Each compartment has to be defined individually, so there is no Cartesian product of graphs at play. The software offers no graphical interface, it is a command line tool, making it error-prone, since the input files take hundreds of lines that have to be written by hand ¹¹. This is similar to what the epidemiologist on our team described. He explained that for custom models, he has to use Matlab and write a great amount of similar lines by hand, which involves copy-pasting. Of course, if the model is not stratified then there would be minimal copy-pasting, but one of our use cases is models with a large amount of stratifications, so this software is not directly usable. The input format for the equations is Lisp-style, like our output format, so there would be minimal implementation difference to support optionally creating .emodl files used by IDM-CMS. There is no notion of configuration in the .emodl format. In our project, we dissociate the model from the parametrization. Like most epidemiological modeling software, IDM-CMS requires an already parameterized model. This makes sense, as IDM-CMS is only a simulator, but it does mean that when creating models it makes it harder to dissociate the model from a specific parametrization.

Since IDM-CMS does not offer a graphical interface and integrates parameters in the model file, it is simply solving a different problem than the one we are tackling. Our platform focuses on modeling expressiveness and dissociating programming from modeling. The simulation capabilities of IDM-CMS may be inspiring for the future of the project but they are out of the scope of this thesis. IDM-CMS is a simple tool and it should be our objective to make ours as simple as possible, as even though they do not solve exactly the same problem, they are still competing technologies.

¹¹<https://github.com/InstituteforDiseaseModeling/IDM-CMS/blob/master/models/idm>

2.8.3 CSID tools

Climate-Sensitive Infectious Disease, CSID, often refers to diseases which can spread to humans from insects, food or water during certain periods of the year depending on meteorological conditions. A study by the Wellcome Institute¹² identifies 37 tools for CSID modeling [34]. The study only identifies the tools and characterizes them based on the information available without using the tool. This study is of great importance for the project, but not necessarily for this thesis. The most interesting project identified in the report is ANOSPEX: a stochastic, spatially explicit model for studying *Anopheles* meta-population dynamics. It is a model of marsh mosquito populations based on climate conditions. In the report it is the second case study, on page 17. This is a typical model the platform might have to support, as most CSID modeling revolves around vector-borne diseases, as indicated by the report, a large portion being *aedes*-borne. *Aedes Aegypti* is a species of mosquito which may carry various diseases, it is known as the yellow fever mosquito. Insect population models might be representable directly in our platform as is. There is no reason why the specific model mentioned above wouldn't be representable using our platform. Most climate conditions in the models are parameters in equations. Evolution from eggs to larvae to pupae are transitions such as rates. Predation, mating & dispersal are also equations based on climate parameters. If we can accommodate integration of various models such as the one mentioned above, by creating adequate features and documentation, we have an opportunity to facilitate reproducibility by supporting all kinds of models in the same graphical notation.

2.8.4 Kendrick

Kendrick is, in its own words, a DSL and simulation platform for epidemiology modeling [35]¹³. This description is an exact match for our project, but Kendrick's DSL is textual, not graphical. Example models are available¹⁴ There are other differences from our platform. Firstly, structure, equations, parameters and simulation details are in the same file. Secondly, stratification seems possible according to the second example model¹⁵ but it isn't so clear¹⁶. Finally, the DSL may be a form of MDE, but there is no use of an MDE framework. The project is implemented in Small-Talk and runs in Pharo¹⁷ using Metacello¹⁸. Much like many projects, it is hard to verify the features presented as the instructions to run are incorrect

¹²<https://wellcome.org/>

¹³<https://github.com/KendrickOrg/kendrick>

¹⁴Kendrick formal-SoC-model m1

¹⁵Kendrick formal-SoC-model m2

¹⁶Kendrick MetaModel

¹⁷<https://pharo.org/>

¹⁸<https://github.com/Metacello/metacello>

and there is no output from Pharo.

2.9 Key Takeaways

These are notes on gathered experience looking into the current state of epidemiological applications, their presentation and documentation. This is subjective but mostly addressed to future researchers collaborating on our project.

2.9.1 The Cost of Unique Features

There are many platforms for epidemiological modeling and some offers functionalities we can't support as easily as they do. Before introducing user-friendly operations for certain features, the cost in future changes must be considered, as well as the attention cost for users who do not use that feature. Already by offering extensibility, we can support most atypical use cases, even incorporating models from other platforms if required.

2.9.2 Generic Platforms

Though most platforms present their work as generic, there are underlying restrictions on the modeling end which restrict the users' capability to model unique behaviour for a given disease or context. There are already platforms offering modeling for specific diseases and we are not directly competing with them. Rather we should aim to keep our work as generic as possible, though it can hinder adoption from certain groups, it won't completely stop any group from adopting our platform. By doing so, we enable not only epidemiological models to be built with our platform but also reasonably similar models, such as ecological models. Wherever you picture the limit of the platforms' capabilities at this time, consider they might just lie further than that. This limit won't be so clear to draw and will most likely manifest as a result of assumptions accidentally embedded in implementation details.

2.9.3 Usability

For our platform to succeed, we should aim for a single installation step. First impressions can be deceiving, but trying a modeling platform and being unable to start, create a project or simulate, is a bad experience. The previously mentioned cost of users' attention is multiplied during the adoption period which is essentially a make or break situation, even for expert audiences such as researchers. There are many modeling platforms and, for most users, no fundamental reasons to pick one over another.

2.9.4 Composability

Typically, it is best to spend most effort on the most important parts of a project, rather than trying to improve less important aspects. For us, this happens to be the composability features which enable model stratification, comorbidity modeling, model merging, etc.

During the first discussions on the project, the epidemiologist clearly explained two important features for his work that could be automated but had not been in other platforms. Those features are stratification, now presented as products, and graph vertex substitution, now presented as sinks and sources of aggregates, which were the main drive for the composable approach. These operations are fundamental and supporting them allows for similar operations to be implementable without change to the metamodel. These components can be useful and reusable for a small development cost. Fostering composability is a key to platform adoption, as it incentivises user-developers to share their work in a reusable manner for future users to directly integrate those components.

CHAPTER 3 METAMODEL

This chapter presents as well as motivates our requirements and design decisions regarding the metamodel. The primary objective of this chapter will be to demonstrate the validity of our metamodel to support compartmental epidemiological models.

The metamodel is the most fundamentally complex piece of our work. Its design guides both composability and extensibility functionalities, which we have identified as a fundamental requirement for an optimal modeling experience.

Metamodel elements presented in this thesis are the result of a cooperation with an epidemiologist over the course of two years. The idea for model composability using groups and products was brought forward by the epidemiologist. The need for composability from the epidemiological modeling community was not confirmed, but it was hypothesized that this work could help solve the identified reproducibility issues.

The input from the epidemiologist that guided the project was minimal, as the main focus of our discussions revolved around epidemiological modeling tasks, not necessarily about the technology. In his view, the platform was to compete with a programming approach to epidemiological modeling, which is less reliable but more flexible. The material provided by the epidemiologist was mostly about the mathematical correctness aspects of stratifying models and graph vertex substitution, both part of the composability model.

We first present our requirements for our metamodel and eventual extensions, followed by the resulting metamodel and finally an example model.

3.1 Metamodel Requirements

The role of the metamodel in our software is to incorporate epidemiology requirements. Our platform is generic, but enables epidemiological modeling by making use of the extensible metamodel. We have drawn epidemiology requirements by following directives from an epidemiologist and exploring epidemiological models used for transmissible infectious diseases.

First, we identify requirements for the base metamodel, the most fundamental level of requirements:

- Support modeling of compartments.
- Support runtime polymorphic compartment alternatives such as products & groups.

- Support modeling of flows.
- Support runtime polymorphic flow alternatives such as batches, rates & contacts.

These requirements implicitly include the following, which we explicitly denote to avoid any confusion:

- Support composability of any compartment identically.
- Support flows referencing any type of compartment identically.

We will motivate support for types of flows and compartments through relevant epidemiological modeling examples. By identifying typical extensions we aim to support, we will draw additional requirements for the base metamodel which will enable seamless integration of most future extensions. The resulting requirements are presented at the end of this section.

3.1.1 Metamodel Extension Requirements

The following presents extensions from which we deduce requirements for the base compartment and flow classes.

Groups

Consider a representation of an SIR model as an array of compartments: $[S, I, R]$. In our platform, this array is an element type called a group. Groups are composed of compartments and flows. Groups can be used in place of a unit compartment to provide more details. For example, a three-element SIR model could be made into a four-element SIR model which accounts for symptoms using a group for I instead of a unit compartment: $[S, I=[I_{Asymptomatic}, I_{Symptomatic}], R]$.

Groups enable graph vertex substitution [36]. By providing groups with sources and sinks, it is possible to connect them to other compartments appropriately. For example, it could be specified that $I_{Asymptomatic}$ is a source and $I_{Symptomatic}$ is a sink. Doing so means that a flow from S to I would then point only to $I_{Asymptomatic}$ while a flow from I to R would point only from $I_{Symptomatic}$. The ability for compartments to identify their sources and sinks to generate appropriate equations is our first metamodel requirement.

Simulation artifacts produced by our models must provide a list of compartments and equations. Compartments such as groups will rely on the artifacts produced by their children. Our models are tree structures, while the produced artifacts are a flattened representation.

Generated compartments will be referred to as physical compartments, as it corresponds to the representation used for simulation. The same applies to flows. The second requirement for our base metamodel is for compartments to produce their physical, or flattened, representation of compartments and equations.

Products

In order to represent model stratification, the simplest approach is to make use of a product compartment type. Products perform a Cartesian product of their children. A product's children are its dimensions.

Typically, Cartesian products produce lists of lists of elements from lists of lists of elements. For example: the product of $[[a, b], [1, 2]]$ is $[[a, 1], [a, 2], [b, 1], [b, 2]]$. In order to keep this property and facilitate other unrelated operations, we choose to represent labels as lists instead of concatenating them. For example, consider two groups A and B with children 1, 2 and 3, 4 respectively. One idea is to represent this physically as $[A1B3, A1B4, A2B3, A2B4]$, but it is hard to make sense of this representation. Instead, by concatenating labels as lists instead of strings, we obtain $[[A, 1, B, 3], [A, 1, B, 4], [A, 2, B, 3], [A, 2, B, 4]]$. This preserves the identity of the original labels and is closer to the mathematical model of a Cartesian product.

Both strings and lists offer similar features such as length, concatenation, etc. but lists of strings are more structured than concatenated strings. This makes it simple to verify if two physical compartments share the same set of labels for example, which is important since it would be a modeling error. Labels as lists of strings is the third requirement for our base metamodel.

We create a fourth requirement, for simplicity, that every compartment from a model that is used to create a physical compartment should have its labels included in this physical compartment. This will allow users and developers to understand or analyse resulting artifacts from their models faster.

Basic Reproduction Number Compartment

It is possible to automatically enable calculation of the basic reproduction number of a model, its R_0 , with an extension. This idea is of course more complex than groups or products, and serves more as a thought experiment to validate the proposed metamodel requirements.

This extension offers a type of compartment composed of one child compartment, a compartmental model which we want to evaluate the R_0 for. The simulation artifacts produced

by this compartment will duplicate the model in two copies, one containing the originally infectious or exposed population, and one containing the susceptible population. By disabling infectiousness of the second copy, all infections are caused by originally infected people, which we can divide by the number of infectious people in the scenario, enabling the automatic calculation of the R_0 .

There are many implementations of this extension which do not break our modeling assumptions and do not require additional requirements. That is to say, creating two copies and disabling the exposure flow for one of the copies is possible given our proposed metamodel. This extension is particularly useful as it is easily reusable and serves a very concrete purpose. It is a good sign that this moderately complex extension does not add further requirements to our metamodel.

Basic Flows

Typical epidemiological models only make use of rates, batches and contacts, which are simple to implement. However, implementing just these three types of flows in a uniform manner does pose problems. Ideally, every equation can be represented in a simple but flexible format for the simulation. The problem is to find a format that works well with the basic flows and supports future types of flows.

Our solution is to represent equations in a Lisp-like syntax, enabling arbitrary mathematical expressions that are simple to parse. Each flow has a source, a destination and a Lisp-like equation. We would represent the contact in a typical SEIR model like so: {from: [S], to: [E], equation: (sumproduct [S] [I])}, omitting parameters for simplicity. Note the use of sumproduct is essentially a dot product but takes as parameters an arbitrary number of vectors of the same dimension, here it is simple to imagine [S] and [I] being single compartments, but in practice there are multiple based on age and other stratifications. Brackets are understood by the simulator to denote the compartment identified by the contained labels, the value of which is to be evaluated at each time step. In the event where [S] or [I] denotes multiple compartments, such as each age group, this evaluates to a vector of population values. The use of Lisp-like makes it simple to parse nested equations, as it does not require operator precedence, unlike regular mathematical expressions.

Most equations require at least one constant to calibrate them. Consider two parameters of a contact, susceptibility of the source compartment and contagiousness of the contact compartment. Including these parameters in the equation produces (sumproduct (* (parameter susceptibility [S]) [S]) (* (parameter infectiousness [I]) [I])). Similarly to the stratification of the populations, if [I] is stratified along age, parameters relying on [I] are stratified. This

means that (parameter infectiousness [I]) has the same number of elements as [I] and so does their product "*", which is not a dot product but rather a Hadamard product, accepting an arbitrary number of vectors of the same dimension.

Knowing where to place the parameters in the equation requires knowledge about what the purpose of the parameter is. For this to be possible, since a metamodeler does not know the intent of every modeler, the metamodel has to offer each parameterization pattern individually. For example, for contacts, there are five parameterization patterns, which will not be covered now. One of these applies for susceptibility and one applies for infectiousness, so the user simply has to identify the parameterization patterns they wish to use.

For the basic equations, we identify two requirements: the first is using a Lisp-like representation and the second is the ability to assign named parameters to patterns.

Complex Flows

Depending on the disease and context, users will want to make use of more complex equations to represent some phenomena. One such example is the modeling of HIV. Being a sexually transmitted disease, it poses modeling concerns that do not appear in most other infectious disease models.

Modeling HIV is particularly difficult as it is important to account for births as well as relationships. Studies also show that modeling relationship behaviours may be important. For example, depending on social context and available treatments, women can contract HIV at a greater rate than men due to abusive relationships acceptance [37].

In order to model relationships, we require new types of flows. The problem at hand is to model relationship creation and, when they come to an end, remove relationship status of both individuals at the same time.

Undoing relationships correctly is simple. For example, consider a *male–female–relationship* compartment with a rate to compartment *single–male* as well as a rate to *single–female*. Assuming both rates use the same equation, relationships will break evenly, placing the same amount of men and women back to their respective single compartment.

The problem lies in producing relationships given a population of men and women. Making use of batches with set amounts is a solution, but it is not as natural as using rates. Consider the following heterosexual relationship example. Using two rates with the same constant, one from male and one from female, but one making use of the male population and the other the female, we end up with an invalid model. Even a slight difference between the male and female population would lead to a disparity where more or less men are entering heterosexual

relationships than women and our model would not make sense. To solve this problem, we introduce cross-rates, a new type of flow which proves useful in different contexts and helps verify the validity of our current requirements.

Cross-Rates

This new type of flow is similar to a rate but does not use its source population in the calculation, making use of another compartment's population instead. Cross-rates describe a flow from A to B with equation $constant * \text{population of } C$, where C is any compartment. In the context of modeling relationships, cross-rates can be used as flows from *Male* and *Female* to *Relationship* with equation $constant * \text{population of } Group[Male, Female]$.

Cross-rates also show usefulness by allowing for simple births modeling. Incorporating births into our epidemiological models is a requirement for modeling HIV and many other diseases, or modeling of infectious insect populations.

In the context of HIV, there are two main reasons for incorporating births. First, the models require incorporating births to accurately model how the disease evolves in our population over long periods of time, since HIV is not curable. Second, HIV presents perinatal transmission, as a fraction of untreated HIV-positive pregnancies result in HIV-positive newborns¹.

In order to model births, we draw from a placeholder compartment which corresponds to future births and transfer them to a newborn compartment. A natural solution is to draw the number of births based on the current demographics by making use of the number of relationships or simply the number of adults, similar to what cross-rates accomplished for relationship modeling.

Cross-rates add no requirements to our current metamodel. This indicates that the simple flow requirements hold for slightly more complex cases but does not necessarily validate it.

3.1.2 Metamodel Requirements Summary

In order to support the base metamodel and previously mentioned extensions, as well as future extensions, we identify the following requirements for the base metamodel:

- Compartments are to:
 - provide their sources and sinks.
 - produce their flattened physical compartments and equations.

¹<https://www.kff.org/global-health-policy/fact-sheet/the-global-hiv-aids-epidemic/>

- Physical compartments are to:
 - be represented using lists of labels.
 - contain each of the labels of the compartments used during their generation.
- Flows are to:
 - support parameterization according to patterns defined by the metamodel.
- Physical flows are to:
 - reference a source and sink physical compartment.
 - use a Lisp-like syntax for their equation.

As we have seen with examples of metamodel extensions, our requirements enable extensions of acceptable complexity.

3.2 Metamodel Implementation

Having detailed our set of types and their use for epidemiological modeling, we now provide an implementation, using the Eclipse Modeling Framework, for our metamodel. The full metamodel is composed of the base metamodel and selected extensions, each of which has a separate diagram, being a different modeling project. The metamodel will first be presented graphically, then some programming implementation details will be included.

3.2.1 EMF Wrappers

Before opening with the metamodel diagram, a caveat has to be settled. The Eclipse Modeling Framework supports metamodel extensibility in a limiting manner. This is best shown through an example.

Consider a metamodel M with extensions $MBox$ and $MPen$.

- Let M declare an abstract class *Object*.
- Let M declare a concrete type *Bag* inheriting from *Object*.
- Let $MBox$ declare a concrete type *Box* inheriting from *Object*.
- Let $MPen$ declare a concrete type *Pen* inheriting from *Object*.
- Let both *Bag* and *Box* be composed of $0..*$ *Objects*.

The expected behaviour is for both *Bag* and *Box* to be allowed to contain *Pens*, but this is not the observed behaviour. The extension mechanism of EMF is for extensions of *M* such as *MBox* and *MPen* to add their *Objects* as alternatives for members of *Bag*. Since *MPen* and *MBox* are unaware of each other, *Boxes* can't contain *Pens*.

The solution to this is to introduce wrappers. By introducing a wrapper such as *ObjectWrapper* in *M*, a concrete type composed of exactly 1 *Object*, the problem can be solved. *Bags* and *Boxes* will now be composed of 0..* *ObjectWrappers*. The extension mechanism now allows for *MBox* and *MPen* to interact. *Boxes* may contain an *ObjectWrapper* from *M* which was extended by *MPen*, allowing for *Boxes* to contain *Pens*.

This is exactly the situation at hand for our metamodels. In the previous example, replace boxes and pens with products and groups and the situation describes the inability for groups and products to contain products or groups, respectively. The use of wrapper will be present in types making use of composition of compartments or flows in the upcoming metamodels.

3.2.2 Metamodel Graphical Implementation

Base Metamodel

The base metamodel, seen in figure 3.1, declares the top level type *Epidemic* as well as the *Compartment* and *Flow* types:

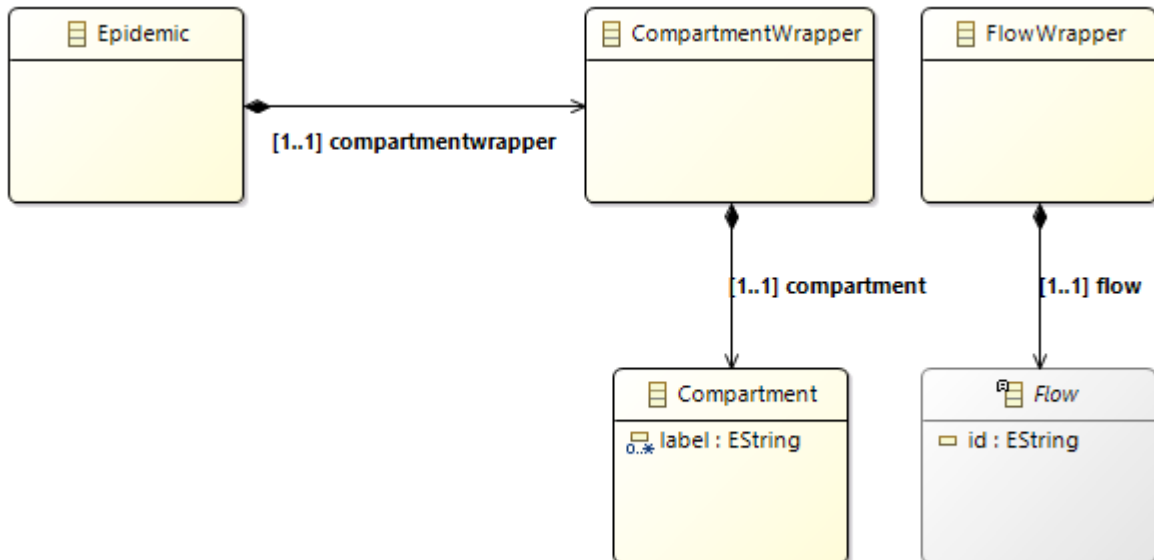


Figure 3.1 Base Metamodel

An epidemic is composed of a single compartment, for example a product or a group. The base *Compartment* type is a unit compartment class which generates a single physical compartment from its' labels. Note that *Compartment* could have been an abstract class and *UnitCompartment* a concrete class, but providing some default functionalities in the compartment class saves for development effort for inheriting types.

Group

Figure 3.2 describes the group metamodel extension.

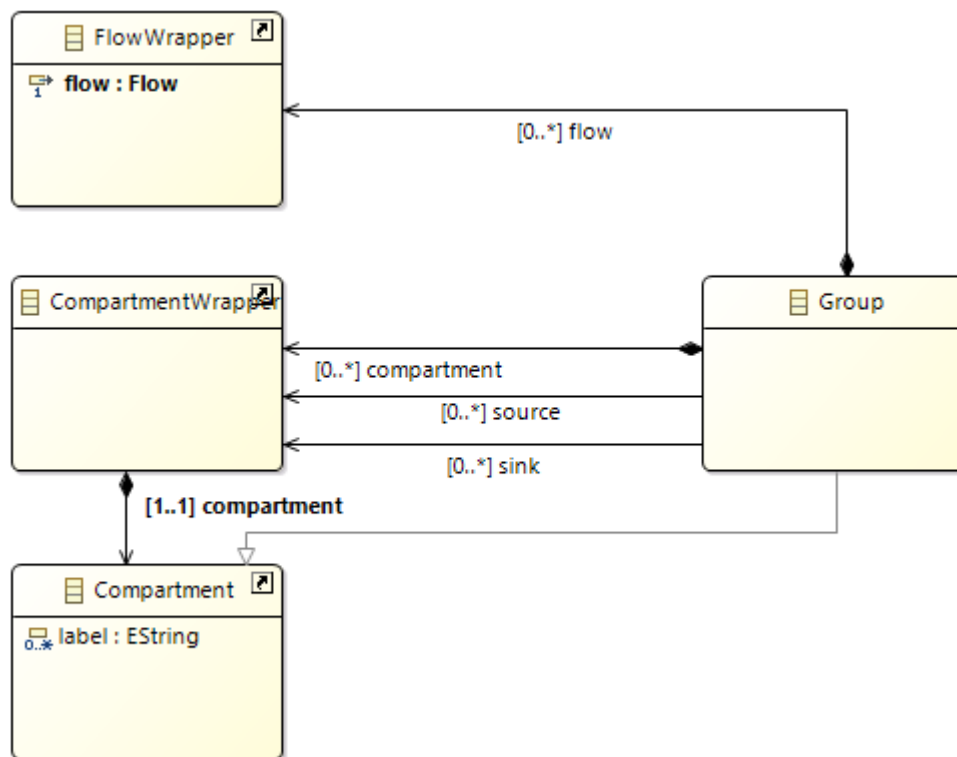


Figure 3.2 Group Metamodel

A group is composed of any number of compartments and flows, and it extends the *Compartment* class. Groups also reference some of their compartments as either *source*, *sink*, both or neither. Not all constraints are representable in the metamodel diagram. For example, though a group could reference any compartment as its sink or source, the referenced compartments must be its direct children.

Product

Figure 3.3 describes the product metamodel extension.

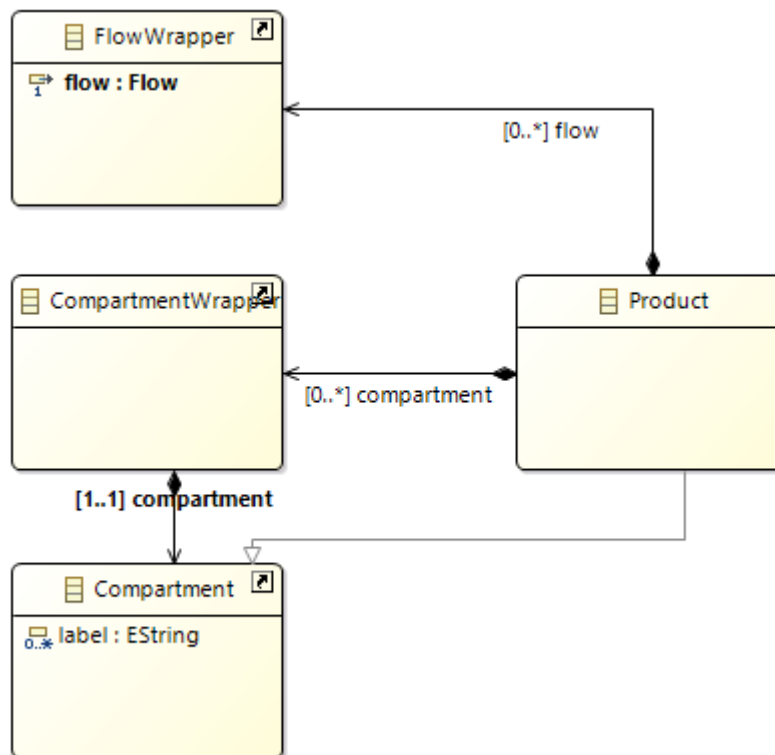


Figure 3.3 Product Metamodel

Products are identical in structure to groups, except for the sources and sinks. Products derive their sources and sinks from the product of the sources and sinks of their children. The ability to place flows in products was not discussed and is not particularly important, it can be ignored. The idea is that a user could place an existing SEIR model in a product and add a flow from R to S, forming a typical SEIRS model. This could be done in groups too, not just products. Peeking inside a group from the outside is not necessarily good or bad, but is up for debate in future versions.

Basic Flows

The basic flows extension, presented in figure 3.4, provides the three basic types of flows, rates, batches and contacts:

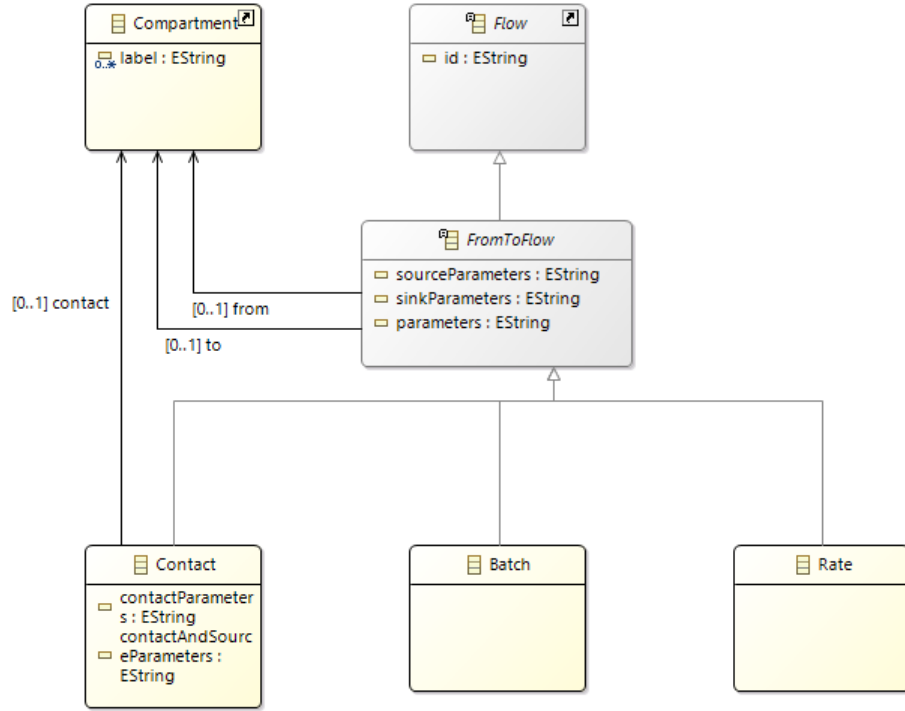


Figure 3.4 Basic Flows Metamodel

By making use of the class *FromToFlow* in the hierarchy some functionalities common to all three flows are easily shared.

3.2.3 Metamodel Programming Implementation

In order to clarify nuances, we present sample algorithms which present basic functionalities in Java-like pseudo code. The following are algorithms used by *Groups* and *Products* to produce artifacts. Note that all `.getCompartment()` methods are generated by EMF from the compositions in the UML diagram being named *compartment*. The version for compartment wrappers returns a compartment while the group and product versions return *EList* of compartment wrappers. *prependSelf* is a utility to add the labels of the current compartment at the beginning of the labels list of the produced physical compartments.

The algorithms to generate sources and sinks are quite similar. Groups gather the source physical compartments of its source compartments, while products produce the Cartesian product of the source physical compartments of their children. The algorithms used to gather equations are nearly identical.

Algorithms used by flows to produce their physical representations are simple but rely on

Algorithm 1 Produce Physical Compartments of Group

```

procedure GETPHYSICALCOMPARTMENTS
  for wrapper  $\in$  this.getCompartment() do
    for pc  $\in$  wrapper.getCompartment().getPhysicalCompartments() do
      yield this.prependSelf(pc)
    end for
  end for
end procedure

```

Algorithm 2 Produce Physical Compartments of Product

```

procedure GETPHYSICALCOMPARTMENTS
  alias PC  $\leftarrow$  PhysicalCompartment
  List<List<PC>> dimensions  $\leftarrow$  []
  for wrapper  $\in$  this.getCompartment() do
    compartment  $\leftarrow$  wrapper.getCompartment()
    List<PC> dimension  $\leftarrow$  compartment.getPhysicalCompartments()
    dimensions.add(dimension)
  end for
  List<List<PC>> product  $\leftarrow$  CartesianProduct(dimensions)
  for List<PC> pcs  $\in$  product do
    PC pc  $\leftarrow$  join(pcs)
    yield this.prependSelf(pc)
  end for
end procedure

```

parameterization patterns which will be covered in the context of the case study.

3.3 Example Model

Figure 3.5 presents an example model which makes use of previously described units, groups, products, rates and contacts:

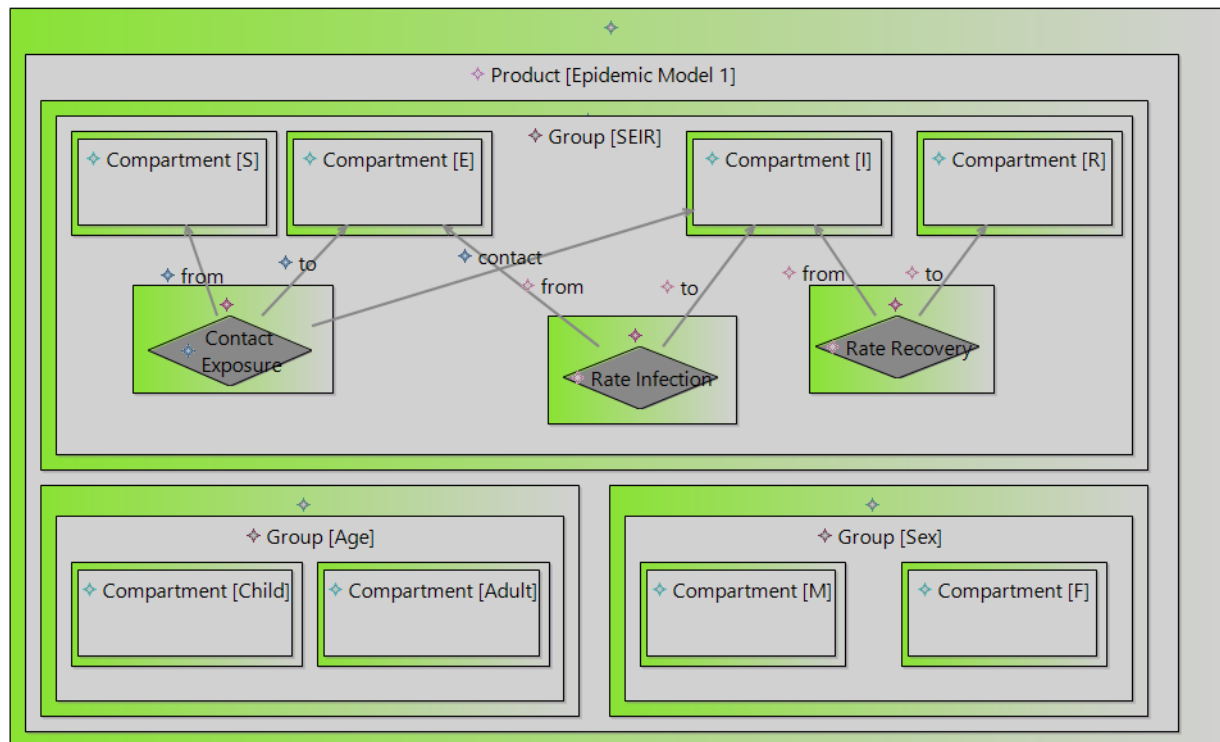


Figure 3.5 Example Model

This model will generate a total of 16 physical compartments, for example: $[S, Child, M]$. Each flow in the SEIR dimension is multiplied for each age and sex stratification, yielding 12 physical flows, for example: Exposure from $[S, Child, M]$ to $[E, Child, M]$.

Keen readers may also have noticed the lack of mortality in the model. Indeed, one of the main epidemiological modeling concerns being mortality, most models will include it. A COVID-19 model will be presented during the case study, section 5.1, along with a discussion regarding mortality modeling in typical models and in our platform.

CHAPTER 4 PLATFORM

This chapter presents requirements and design of the platform features not directly related to metamodeling or modeling. It covers subjects such as model comparison, the graphical interface and model parameterization and simulation. This chapter serves as an overview of the context in which developers and users can think of the platform, abstracting epidemiology and metamodeling concerns.

4.1 Platform Requirements

Understanding how and why epidemiologists will use our platform is the first step in order to design it. With the help of an epidemiologist, we identified atomic use cases for our platform to support, presented as user stories in Table 4.1.

The present thesis concerns itself with user stories 1-5, 14 and 20 for creating and managing epidemiological models, as well as 22, 23 and 24 which enable metamodeling for platform users.

We identify platform features required to fulfill the identified user stories:

- Base metamodel type system{1, 22, 23}
- Graphical modeling interface for model creation{1} and modification{5}
- Model comparison{4}.
- Runtime metamodel extensibility using plugins{14, 24}

Our requirements and metamodel seem to enable the modeling of any infectious disease, but are incomplete. Unforeseen concerns inevitably reveal themselves during design and implementation. In order to create a platform, some design and implementation decisions have to be made, as there is a gap to fill between the high level requirements and the implementation. The following section describes the platform, bridging this gap, it describes how users make use of our metamodel in practice.

Table 4.1 Infectious Disease Modeling User Stories.

As a modeler, I want to:		
01	create models.	
02	store models,	to access them over time.
03	share models,	to collaborate with other modelers.
04	compare models,	to automatically list differences.
05	modify existing models,	to improve them.
06	add data sources to a model,	for the purpose of calibration.
07	calibrate models,	to evaluate fitness of model results to available data.
08	inspect the parameter values from calibration,	to verify/validate calibration results.
09	simulate scenarios of interest,	to answer questions of analysts.
10	make reports of scenario results,	to present to analysts.
11	analyze the behavior of the model,	to verify/validate scenario results (i.e. sensitivity analysis).
12	recalibrate models,	to take into account updated data sources.
13	document the models (structure, parameters, calibrations, scenarios, results),	to make reports and/or publications.
14	make use of metamodel extensions,	to simplify or accelerate the modeling task.
As an analyst, I want to:		
15	define scenarios of interest	
16	consult scenario results or query new results by running new simulations	to answer research questions and inform public health policy.
17	view the model's structure, parameters, and calibrations,	to elicit new scenarios and to ask questions about the model to modelers (e.g., flagging issues).
As an auditor, I want to:		
18	consult scenario results or query new results,	to understand the model's behavior and/or test the model.
19	view the model's structure, parameters, and calibrations,	to verify/validate the model.
20	run a simulation,	to verify/validate reported results.
21	review the documentation of the models	to understand their structure, parameters, calibrations, scenarios, and results.
As a metamodeler, I want to:		
22	implement new types of equations,	to represent real-world behaviour of a disease or population.
23	implement new types of compartments,	to represent an existing population classification.
24	share my metamodel extensions,	to facilitate collaboration and reproducibility.

4.2 Platform Design

4.2.1 Project Creation

The first step for a user to access the epidemiological modeling environment is to create a project. Each project contains an epidemiological model as well as metadata describing extensions and settings. Deploying a platform inside an existing environment simplifies this task and offers it intuitively to the user. For example, with Eclipse, extending the project creation wizard to offer a type of project is the appropriate choice.

By making use of the Eclipse plugin extension points, it is simple to implement our project creation window. The project creation window is available through navigating the typical project creation process, as seen in figures 4.1 and 4.2.

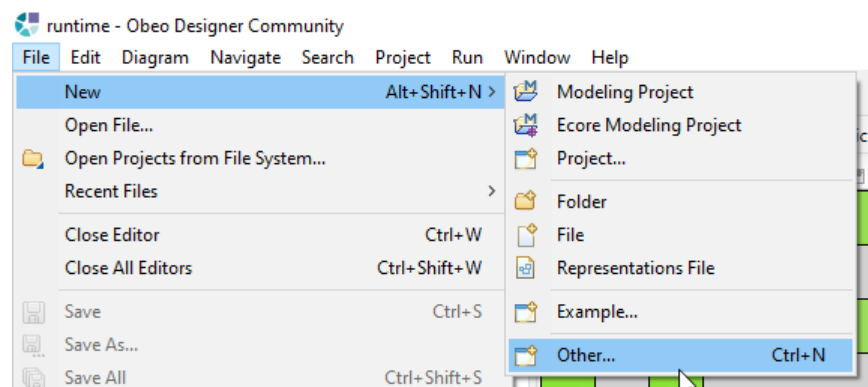


Figure 4.1 Normal Project Creation

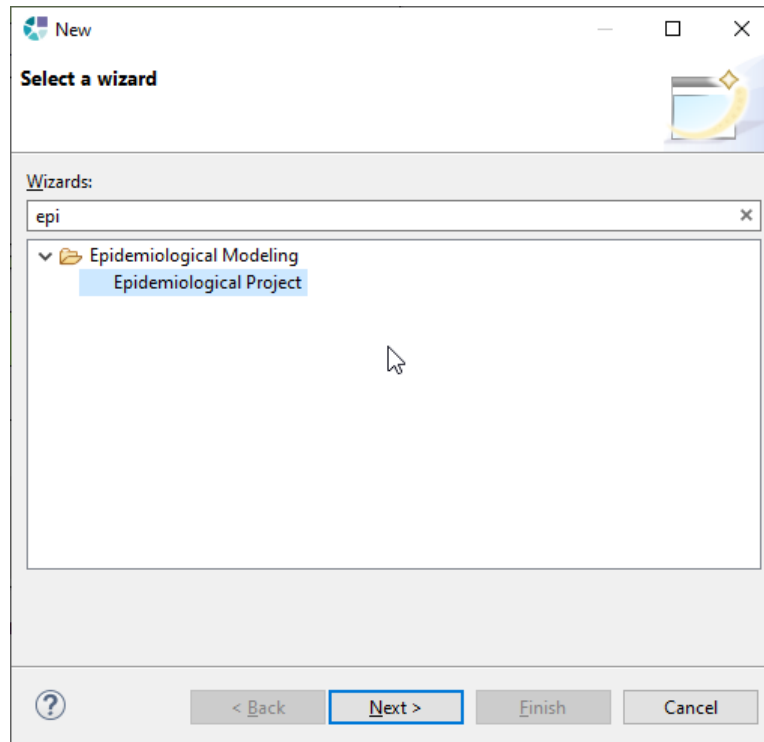


Figure 4.2 Epidemiological Project Selection

4.2.2 Metamodel Plugin Selection

Metamodel extensions can come in the form of plugins. Using a programming language such as Java for the platform is recommended in order to easily manage plugins. Exporting and importing Java Archives, JARs, is a simple process and can be diagnosed in failure cases.

Given the simple nature of the proposed extension mechanism, users may directly handle the plugin files, as there is no installation process. It should be possible for the user to specify once where all their plugins are located, for example, by using an environment variable to designate a directory. Placing the plugins in the correct directory renders them available to the plugin selection feature.

During project creation, or afterwards, users select metamodel extensions for their project. Paying attention during selection is valuable, as selecting the minimal extension subset makes modeling simpler. Users are to be encouraged to select the appropriate plugins for their projects by following examples and guidelines. Figure 4.3 displays the plugin selection window during project creation.

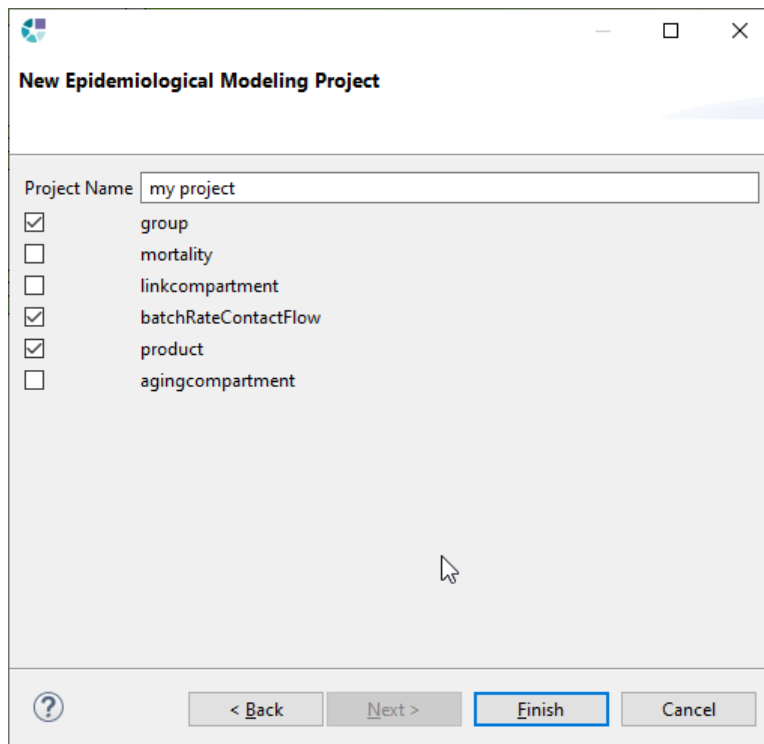


Figure 4.3 Plugin Selection

Plugins are gathered during launch using the directory specified by the user which contains the JARs. Each JAR may contain multiple *EPackages*, essentially metamodeling projects. Each *EPackages* is considered to be a plugin. They are added to the *EPackage.Registry* used by EMF. Types declared in the selected *EPackages* are available to the user during modeling.

While modeling, classes which extend *Compartment* or *Flow* are offered to the user when adding new elements of these types to a model. For example, when clicking to add a flow to a group, this is performed to find all concrete flows enabled for this project.

4.2.3 Graphical Interface

Compartments are represented using rectangles which contain flows and compartments. Compartments and flows have their name and type displayed within their shape. Flows are not arrows, but rather diamonds with labeled arrows pointing from it to compartments. Each line pointing from a flow to a compartment is labeled according to its role. For example, a contact *C* has three arrows labeled "from", "to" and "contact" and on the diamond is displayed "Contact *C*".

Clicking on a compartment or flow offers a view of its properties, such as any value that is not present or modifiable in the graphical view. The label is an example of such a property. For more details, extra options can be accessed. Groups, for example, offer a form of toggle for each of their children marking them as sink, source, neither or both. This is not visible as a property but rather requires opening it in a popup, this popup is presented during the case study in figure 5.4.

Compartments and flows should offer a visual warning when they are incorrectly configured, this is not implemented yet in our platform. The ability to generate simulation artifacts from a model with at least one invalid element should be disabled or necessarily produce an error, which is the case with our implementation. This can happen for example when removing references from a flow to a compartment and not pointing it to a new one, or when removing all the labels from a compartment. There are other invalid model situations, such as making use of the same labels twice on different compartments, these do not necessarily have to be caught by the GUI.

Designing a GUI for an extensible type system requires the use of some meta-programming. Using languages which support reflection or by making use of a metamodel, it is possible to draw some information about the properties of an object which should be offered in the GUI. Otherwise, it is required to resort to a programming approach in which the developer of an extension has implement the rendering of their types in the GUI.

The graphical interface is made available using a Sirius viewpoint definition project. It allows us to define rules for presenting model element hierarchies and relationships. Specifically, we define two important rules, one for compartments and one for flows. Sirius allows recursive definitions using the *reusedcontainermappings* property, which we can apply to compartments so that their children compartments recursively display their children.

A limitation from the combination of using Sirius and supporting open sets of types is that rendering types which are not compartments or flows is not supported. Our platform allows any model to be loaded, even if some elements are from an entirely different hierarchy. The Sirius viewpoint would not know how to display these objects, so they may only be accessible through properties of the object they belong to, through a popup implemented by the developer.

Finally, implementing buttons and manipulations for the model is quite tedious and requires multiple steps, such as code in each type, Java external actions definitions, hooks in the Sirius definition, references to the actions in the plugin, etc. This is an area of improvement for future revisions of our platform, since it is likely easier to implement these features by making better use of advanced Sirius features.

4.2.4 Model Comparison

Comparing models with their previous versions or with similar models helps detect errors and understand model behaviour. It is an important task to increase model reproducibility as it enables users to definitely know when two models present differences. For model versioning it accelerates common tasks such as merging or extracting commonalities.

For our purposes, model comparison operates at the structural level but provides information aggregating the physical representations produced by the model. In other words, if two models are different but produce the same representation, this will be clear to the user. Similarly, if two models are structurally identical but a property differs which yields a different physical representation, this will also be presented to the user.

Metamodelers may find these features helpful in the event where they want to develop a metamodel extension. By creating a reference model which produces the desired output and comparing it with the result of a model making use of the extension, they can automatically validate the output of the extension.

Model comparisons relies on identifying a difference as either retyping, renaming, moving, adding or deleting, then synthesizing the differences for the user in their most useful form. Some model difference patterns can be synthesized into more easily understood ones, such as presenting matching added and deleted elements as moves.

In order to implement model comparison for an extensible type system, while providing relevant comparison results, extension developers have to add their touch. Specifically, each type overrides a comparison method which operates for same type comparison, this is where differences can be synthesized. As for comparison of elements of different types, it is essentially impossible to provide useful structural information, as neither type necessarily knows about the other. In this situation, the comparison operates strictly on the generated artifacts of each element to extract differences or commonalities.

The result of a comparison of two objects should be composable since the compartments are composable. In order to compare two objects of the same type, they compare their matching children pair-wise, making use of their comparison results. This is a complex problem that is potentially not well solved, but relies on principles similar to the ones used for artifact generation. It is an area for improvement.

Matching

The first phase of the comparison algorithm is matching. In this phase, we ignore structure and look for identically labelled elements. We are allowed to do so for our type of model, as

label lists are similar to unique identifiers. This assumption is always respected since artifact generation relies on this property. This property could be relaxed but it currently also serves as protection against user errors.

Algorithm 3 SameUniqueLabelMatch

```

res ← new MatchResult()
uniqueLabels ← intersection(unique labels of model 1, unique labels of model 2)
for each Compartment c1 ∈ model 1 do
  for each Compartment c2 ∈ model 2 do
    if res.find(c1) is empty and res.find(c2) is empty and intersection(c1.labels,
c2.labels, uniqueLabels).size() > 0 then
      Add Match(c1, c2) to res
    end if
  end for
end for
for each Match match ∈ res.matches do
  leftParent ← m.left.eContainer()
  rightParent ← m.right.eContainer()
  if leftParent is a Compartment and leftParent same class as rightParent and
leftParent not matched and rightParent not matched then
    Add Match(leftParent, rightParent) to res
  end if
end for
Return res

```

Any matching algorithm choice has to be done considering trade-offs. We will present two limitations of our simple algorithm. First, consider the loops through compartments of both models. It is possible that two compartments of a model, contain two different unique labels both found in a compartment of the other model. This would be most peculiar but nothing stops the user from placing two unique labels on the same compartment. Our algorithm matches only the first, it simply depends on the order they appear in the hierarchy we iterate. Second, consider the loop through matches attempting to match their parents. Consider two compartments *A1* and *A2* with a parent *A* in one model, along with compartments *B* and *C* in the other model. It is possible that *A1* matches *B* and *A2* matches *C*, creating a conflict between matching *A* with the parent of either *B* or *C*. Once again, our algorithm will match only the first one.

Solutions can be explored, for example, in the second scenario, by matching *A* with the first common ancestor of *B* and *C*. This is unlikely to work, as it will create more problematic solutions, such as the event where this common ancestor has already been matched with another compartment. This algorithm is easy to understand and produces intuitive results,

but may be improved in the future.

Differentiating

The second phase of the comparison is differentiating, in which matched elements are to produce a comprehensive overview of their commonalities and differences.

Algorithm 4 Differentiating

```

highestLevelMatch  $\leftarrow$  first match in matches
diffs  $\leftarrow$  {highestLevelMatch.first.compare(highestLevelMatch.second)}
for each Match match  $\in$  matches do
    if match not yet accounted for then
        diffs.add(match.first.compare(match.second))
    end if
end for
for each Difference d  $\in$  diffs do
    for each Compartment c  $\in$  d.additions do
        if c is matched then
            Set match as move
        end if
    end for
end for
Return diffs

```

Each matched compartment is necessarily compared, but since aggregate compartments of the same type compare their children, they are marked as *accountedfor* and synthesized as part of their parent comparison. This allows us to gather the differences more tightly, especially for presenting them to the user. The partial ordering produced by the matching algorithm allows us to iterate directly through matches and always find a parent match before their children, enabling the *accountedfor* process. Once the differentiation is done, moves are also synthesized using matching additions and subtractions.

Finally, the implementation details for each element lie in their implementation of the compare method. Not every type has to offer an implementation, assuming they follow basic structural requirements which are verified using the metamodel. Specifically, metamodelers are not required to provide a comparison implementation for types which make use of a single compartment and flow feature. A feature is a modeling term used in EMF designating a composition or references to a type of value. In other words, a compartment type which has a single list of compartments as its children will be able to use the default comparison.

4.2.5 Parameterization

In order to use a model for simulation, it is necessary for the user to provide values for each equation parameter and original compartment population. Our solution offers parameterization in the same environment as the simulation, in a Python notebook, as both tasks are close to data science. This enables a very short feedback loop between verifying a model behaviour in the simulation and making changes to the parameters.

Equations inside products with many dimensions produce a large amount of parameters that are likely closely related. Our solution offers an API for selecting parameters using operand criteria to set all relevant instances at once by providing a tensor of values. Generic Python functions for manipulating parameters and scenario values allow for parameterizing a model in a reasonable time. If a parameterization call is invalid it is flagged right away, such as providing an incorrect amount of values for a selection of parameters.

Scenarios

A scenario is a distribution of the population in compartments at the beginning of a simulation. For example, a COVID-19 simulation for Canada requires a scenario with around 38 million people placed in the compartments, while a simulation for the united states requires around 330 million, but also the age and comorbidities distributions. Much of the code for scenarios is specific to the geographical area of the studied population, unlike parameters in equations.

Parameters

Parameters refer to coefficients used in equations, which should be engineered to scale accordingly for different models. For example, it should be possible to re-use equations from a Canadian model in a United-States model. If this is not possible, it may be possible to use the same parameterization code for both models, which is the next best alternative for collaboration.

As mentioned previously, the current parameterization solution is available through Python, to be used in a notebook for quick development iterations. Users may use

```
select_parameter_string_equals(0,'contact - mixing')
```

to select all parameters for which the first argument is *contact - mixing* in order to provide them with a value. Criteria can be chained, as to further specify the search. For example,

in order to set this contact mixing parameter to zero for contacts with deceased people, add the criteria

```
select_parameter_compartment_contains(3, 'Deceased')
```

to the previous selection.

Parameters always have at least 2 arguments, the first is their name, such as *contact – mixing* and the second is the name of the equations they are used in. More arguments are required to differentiate parameters used in variations of an equation, such as contacts between different age groups for example. Specifically for the contact, the third argument is the source compartment and the fourth is the contact compartment, which explains the index 3 in our example. This process is not necessarily intuitive or perfectly safe, but it was the simplest complete solution to produce parameterization for large models and adequately supports our case study in the next chapter.

4.2.6 Simulation

Simulation refers to the application of equations over time, making use of equations, parameters and a scenario. Typically it is used to generate graphs used for estimating hospitalization, mortality, peak cases, etc. The output of a simulation are values for each compartment at every time step. A single simulation can then be used to produce multiple graphs of different aspects.

Implementing simulation can be complex, as efficiency is important and the extensible type system offers the possibility for various flows to be used, meaning some generic format for equations is required. The presented solution makes use of a Lisp-like syntax to easily create the equations from modeling and parse them for the simulation. It also integrates easily with the composability model used by the metamodel.

Before simulation, a compilation phase where the equations are parsed and used to generate Python functions is performed. Each equation will be evaluated thousands of times during simulation, so any time lost to compilation can be repaid many times over by saving simulation time.

The compilation phase applies some simple optimizations. For example, compartments are placed into an array, making use of indices rather than labels for accessing the values. There are no string comparisons during simulation. Similarly for parameters, where the corresponding value is found during compilation in the parameterization file and inlined in the equation for simulation.

To simulate, all the user needs to do is specify a time span, a set of points in time the

simulation will provide values for. This time series does not require regular steps and the steps do not necessarily dictate simulation detail. For example, the user may ask for daily steps for a year, followed by weekly steps for 4 years, but the simulation will step multiple times per day independently of the time span, in order to produce accurate results.

Simulation uses an ODE solver, such as the Fourth Order Runge-Kutta method, or approximates by making use only of the current derivative at a given time. The accuracy increase by making use of higher order derivatives instead of only the first derivative, at the cost of increased simulation time. The current simulation environment supports both approaches with a toggle.

According to our experiments, for a model with a large amount of compartments, the cost of using an ODE is too large. Population changes to any compartment value affects hundreds of equations, leading to confusion for the ODE as to the evaluation of the higher order derivatives, which makes it collapse to very short time steps. The current implementation makes use of a very naive approach in which the user provides a tolerance which each compartment change between two steps must conform. If the tolerance is not reached between a step from t_1 to t_2 , we recursively call the simulation with 10 steps $[t_1, t_1 + 0.1 * t_2, ..., t_1 + 0.9 * t_2, t_2]$, which may or may not further divide the steps based on tolerance.

Finally, we offer a functionality to plot values easily by using the labels found in the model. For example, `plot_by_labels(['Alive'], ['Deceased'], ['I - TB'])` plots two curves: *Alive* and *Deceased*, for all *I - TB*, compartments of people infected with tuberculosis.

CHAPTER 5 PLATFORM USAGE & EVALUATION

In order to present the platform in a deliberate manner, an example modeling task is performed. The example is a case study which consists in implementing a published COVID-19 model using our platform and evaluating the experience. This case study serves as grounds for an evaluation of the work presented in this thesis.

5.1 COVID-19 Case Study

We will reproduce a model from 2020 presented in the Canadian Medical Association Journal [38]. This model was not cherry-picked, but it does fall into the family of typical transmissible infectious disease models the platform is built to support.

The authors evaluate the effectiveness of physical-distancing interventions in Canada. They evaluate how different interventions could be used to control the coronavirus disease. This impact is present both in terms of number of affected individuals and in stress on the health care system. The model uses data from the Ontario population. Their objective was to identify "strategies that keep the number of projected severe cases (hospital and ICU admissions) within a range that would not overwhelm the Ontario health care system" [38].

5.1.1 Model Presentation

Figure 5.1 displays the original figure from the article, Model structure of COVID-19 transmission [38, Figure 1].

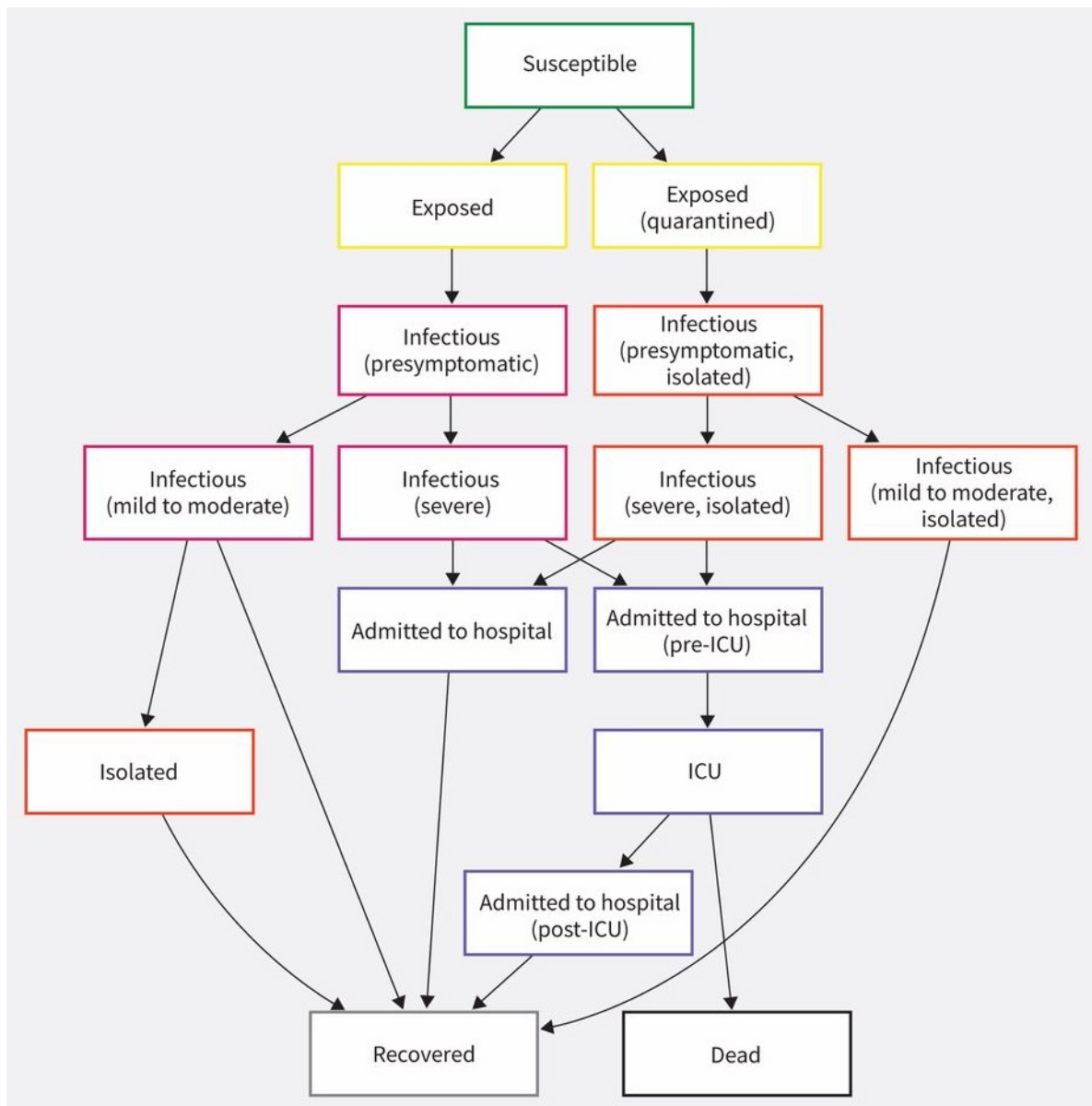


Figure 5.1 Original authors' model representation

Further information in the article notes that this model is stratified along age and comorbidity, which will be used in equations but are absent in the graphical notation. The stratifications used by the researchers are a consequence of their available data. The population age data uses 20 intervals of 5 years and one for 100 years old or older. The contact mixing data uses 14 intervals of 5 years and one for 70 or older.

5.1.2 Platform Workflow

Subsequent sections present in detail how the case study model is created in our platform. This section presents an overview in order to facilitate understanding of the following sections.

A modeling project is created in Eclipse, in which the population structure is created to match as closely as possible the studied model. This entails using products and groups where appropriate to minimize the number of components, as is a key idea for our platform, but staying true to the studied model. If possible, no flows are added during the structural aspect of modeling, and appropriate names are used for each component in order to identify mistakes as early as possible in a modeling process, especially for new models rather than case studies.

Select sinks and sources for each group which has specific sinks and sources, otherwise the platform assumes all options of a group are valid sinks and sources. Failing to select appropriate sinks and sources will generate unwanted equations, so it should be apparent during parameterization if this step was forgotten. Special extensions may require other kinds of parameterization in order to generate the desired output.

Flows are added to the structure in order to match the behaviour of the studied model. This is especially easy if the studied model only makes use of rates and contacts, otherwise the use of metamodel extensions may be required. Each flow requires selecting the appropriate parameterization patterns, in order to add parameters such as susceptibility, infectiousness, a contact matrix, etc. Failing to select any parameterization pattern will produce an error, while selecting the incorrect one should be apparent during parameterization.

Finally, the simulation artifacts can be generated from Eclipse and opened in a Python environment. The Python environment produces errors if the provided parameters do not fulfill the equations and offers an API for selecting and providing values for vectors of related parameters. The same is required for providing compartment population values for the beginning of the simulation. Once the parameterization code is complete, the artifacts and parameterization file are handed to the simulation Python notebook. A simulation output can be saved into a file, or used directly to generate figures such as curves of infections over time by age, by comorbidity, etc.

5.1.3 Modeling Using Our Platform

In order to represent the disease model in our platform, there are two approaches. The first, the naive approach, is to recreate the 16 unit compartments and 21 flows. This is a good way to recreate the same model visually by arranging the model in the same fashion. However,

this approach is only valid for this copying process and is not necessarily natural, nor does it favor modification. The second approach, which we recommend, is to make use of groups and products to reduce the amount of flows and unit compartments, leading to a simpler representation and increasing isolation of concerns.

First, we look at the naming and coloring of the elements in the original figure from the article. For example, we can see that orange refers to isolated, while purple refers to not isolated. Susceptible, exposed, hospitalized and recovered are also color coded.

We begin by creating a group to place our elements into, then place unit compartments for S, R and Dead. E and I share commonalities, specifically, they are both stratified along isolation status. We will create a product to hold two groups: isolation and E-I. The isolation group simply has a isolated and non-isolated unit compartments. E-I has two groups, E and I, which have their purple and orange compartments from the original figure, so E is a compartment but I is a group containing presymptomatic, mild and severe.

The researchers responsible for the studied model made the decision not to include "within-hospital transmission cycles" [38]. For this reason, we can leave hospitalization out of the infectious group as they will not be used as contacts for transmission. Hospitalization will be a group with units for no-ICU, pre-ICU, ICU and post-ICU.

Let us present the structure so far, see figure 5.2.

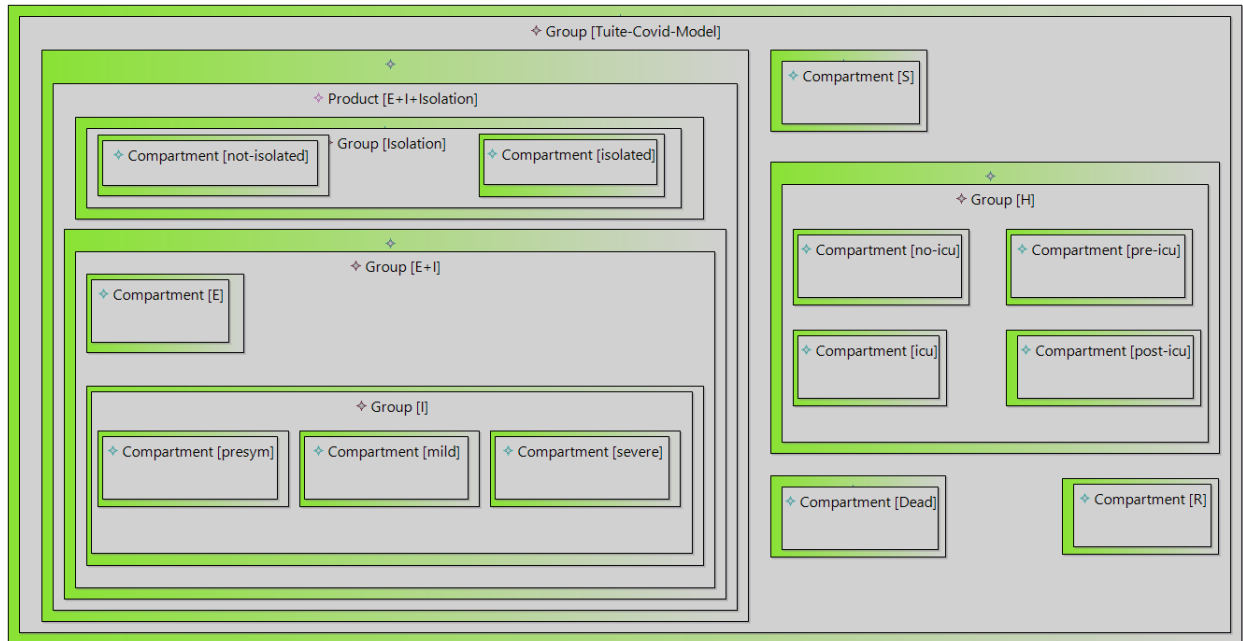


Figure 5.2 Model Structure

After modeling the structural aspect of the model, it is time to model the behavioural aspect, by using flows. The original model contains 21 flows, but by using flows on composites, we can reduce this amount. As a model's size increases, applying flows to structures instead of individual components saves time and complexity. It also dissociates a composite's contents from the flows, allowing for inner structure modification without change to the flows. This is visually apparent in the resulting model, see figure 5.3.

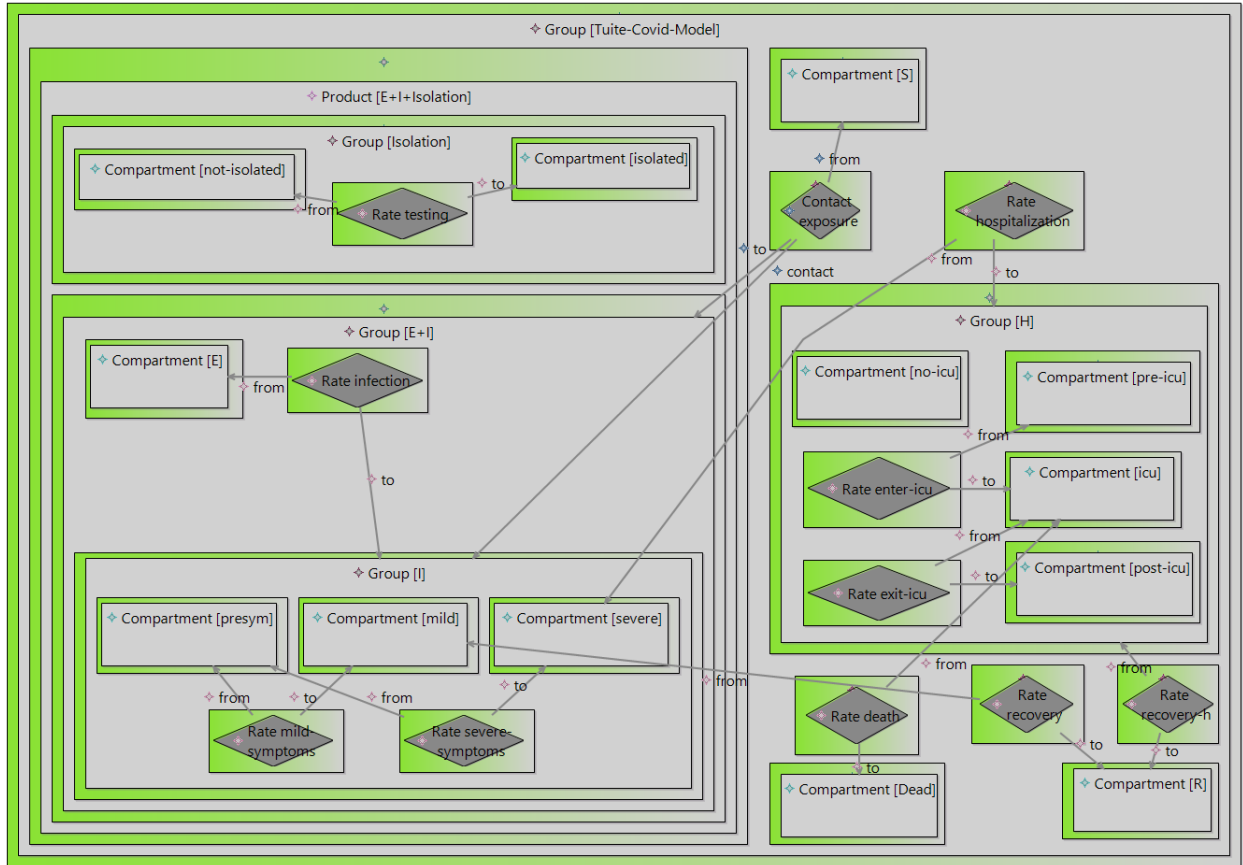


Figure 5.3 Model Structure and Flows

There are now 11 flows instead of 21. Flows in our platform are named diamonds with named reference, which may help clarify the model. For example, the authors mention: "Testing was assumed to move individuals with nonsevere symptoms from the infectious to isolated compartments" [38], while in our model we can simply name this flow "testing".

It is required to select sources and sinks where appropriate, for example the hospitalization group, see figure 5.4. This will allow our software to only generate inputs to the sources and only output from the sinks.

Compartment	Is Source	Is Sink	Delete
[no-icu]	☒	☒	Delete [no-icu]
[icu]	☐	☐	Delete [icu]
[pre-icu]	☒	☐	Delete [pre-icu]
[post-icu]	☐	☒	Delete [post-icu]
			Add Child

Figure 5.4 Source and Sink Selection

As you can see from the sink selection, *ICU* is not a sink. This won't stop our death flow to move population from *ICU* to *dead*, as that flow uses *ICU* directly, but will stop the move from *ICU* to *recovered*, as the *recovery* – *h* flow uses *hospitalization*. This is not typically how we recommend implementing mortality in our platform, but it more closely matches the original model structure.

The final step is model stratification along age and comorbidity, which is simply done by placing the model in a product along with the age dimension and the comorbidity dimension. As seen in the original model, stratification along age and comorbidity is not represented graphically. In our platform, no modeling concern is above being represented graphically.

For stratification, we will leave the first model intact, differentiating the researchers' model structure from the stratifying to the available Canadian data. We create a simple model of a product between age groups available in our data, the researchers' model structure, which is a link, and a comorbidity option, shown in 5.5.

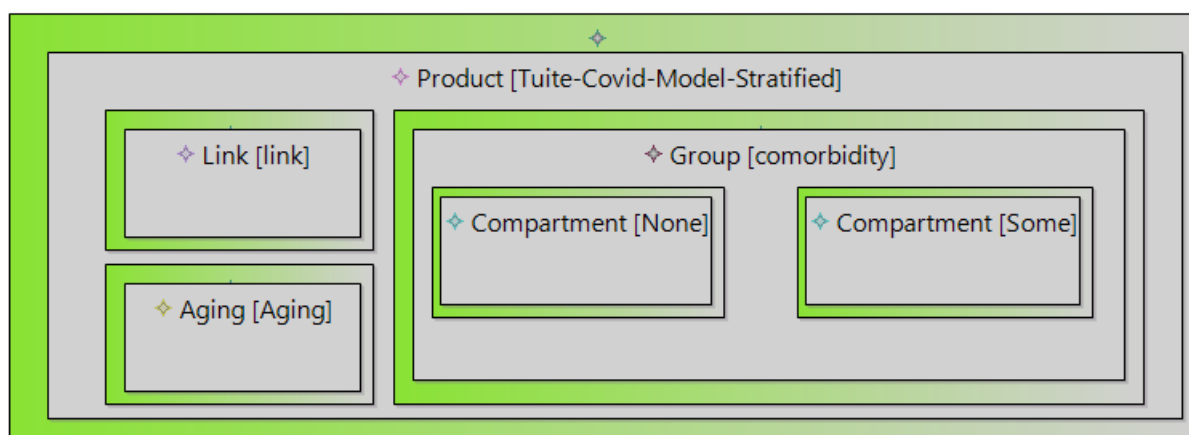


Figure 5.5 Stratified Model

The Link and Aging compartments are simple metamodel extensions. Link allows us to reference compartments from other model, here we reference the top level element from the previous model. Aging allows us to provide a list of age groups as a string, in this case, for the Canadian health data, we provide "0-4,5-9,10-14,15-19,20-24,25-29,30-34,35-39,40-44,45-49,50-54,55-59,60-64,65-69,70+", which generates compartments as well as rates between the consecutive age groups.

Our model is nearly exactly identical to the model we are reproducing, but does present differences. These differences are negligible in the context of this thesis. The first difference is the absence of births, quarantine and interventions. Equations for births and quarantine are trivial as they follow the same pattern as other equations found in the model. They were omitted as they did not appear to be present when reading the article, but were found later in the code base. Interventions are quite particular as they affect a large part of the model periodically when enabled through the dynamic interventions option found in the R code. The logic was not very straight forward so we opted to ignore interventions. For our platform, we recommend creating two copies of the model parameters, one for the normal behaviour and one for the behaviour during interventions. The task to reproduce the model was quite complex because of the intervention mechanism and retrieving and computing of parameters at runtime.

At this point we have produced an adequate representation of the structure of the model along with flows but still require one more modeling. The final step is required in order to provide information to the artifacts to accurately parameterize our equations. Each type of equation, say rates or contacts, conform to specific templates which are parameterized in a structured and predictable manner. Our platform offers each parameterization pattern separately, but at least one must be used. The same patterns appear implicitly or explicitly in every epidemiological model.

We will go over the five patterns which are available for rates and contacts.

1. Single-value: Used to multiply all equations generated from a flow with the same value.
Example: The flow from *ICU* to *post-ICU* has a rate of $1/21$. There are 32 instances of this flow, 16 age stratifications and 2 comorbidity stratifications, but all 32 flows have the same rate.
2. Multi-valued (source): Used to multiply each generated equation with a specific value.
Example: The flow from *ICU* to *Dead* once again has 32 instances, but they each have a different rate based on age and comorbidity.
3. Multi-valued (sink): Used to weigh the outputs of an equation which points to multi-

ple compartments. Example: The flow from *severe* to *H* leads population into either *no-ICU* or *pre-ICU*. Typically, the sum of the sink parameters will be 1 as to simplify reasoning about the overall flow's throughput and read the values like probabilities. In this case, there are 32 age/comorbidity combinations. The parameter used for weighing flows pointing to *pre-ICU* has 32 values. The parameter used for weighing flows pointing to *no-ICU* is defined as a complement so that their sum is a vector of 32 ones.

4. Multi-valued (contact): Used to multiply each contact population with a specific value. Example (not in this model): Providing an *infectiousness* parameter to a contact equation to weigh each infectious compartment. Equation $S_i * sum(I)$ becomes $S_i * sum(infectiousness * I)$.
5. Multi-valued (contact & source): Used to multiply each contact population with a specific value for each source. Example: Contact-mixing. This model and many others make use of a contact-matrix built from a large population study. A given value in the matrix describes how many daily contacts an average person of a certain age group has with another group. A row describes the contacts of a certain age group with all the others, which in our case would be a 16 long vector, or 32 once stratified with comorbidity. The strength of the exposure is weighed with the sum of the element wise multiplication of the appropriate row of the contact matrix and the populations of I compartments. Equation $S_i * sum(I)$ becomes $S_i * sum(contact_i * I)$. In the infectiousness example, the infectiousness values are shared. Here, each equation uses the contact matrix row for its population.

In the modeling interface, the modeler may specify a parameter name for a pattern he wishes to use, see figure 5.6 depicting the usage of two patterns for the hospitalization flow.

id : EString:	?	hospitalization
from : Compartment:	?	◆ Compartment
to : Compartment:	?	◆ Group
sourceParameters : EString:	?	hospitalization-parameter
sinkParameters : EString:	?	p-icu
parameters : EString:	?	

Figure 5.6 Parameter Patterns Usage

The *sourceParameter* and *sinkParameter* are for the two previously described multi-valued parameter patterns. The empty *parameters* field is for a single value parameter but is not used here. It should be noted that each parameter pattern is composable and placing comma separated parameter names will generate a valid equation template featuring each parameter with the appropriate mathematical expansion of the term. For example, we can combine a contact parameter and a contact & source parameter from our examples to generate $S_i * \text{sum}(\text{infectiousness} * \text{contact}_i * I)$. These compositions are automatic whenever patterns are used in conjunction.

5.1.4 Model Parameterization

The final step before running the simulation is to interface with our Python helpers to set the values of the parameters. Figure 5.7 displays the code for the hospitalization flow.

```

110 def setup_parameters_hospitalization(parameters):
111     p_icu = params.p_icu
112     p_no_icu = np.ones(32) - p_icu
113     select(parameters, [
114         select_parameter_string_equals(0, 'hospitalization-parameter')
115     ], N_AGE_GROUPS * N_COMORBIDITY * N_ISO).set(1 / params.inf_period_severe)
116     select(parameters, [
117         select_parameter_string_equals(0, 'p-icu'),
118         select_parameter_compartment_contains(2, 'no-icu')
119     ], N_AGE_GROUPS * N_COMORBIDITY).set_array(p_no_icu)
120     select(parameters, [
121         select_parameter_string_equals(0, 'p-icu'),
122         select_parameter_compartment_contains(2, 'pre-icu')
123     ], N_AGE_GROUPS * N_COMORBIDITY).set_array(p_icu)

```

Figure 5.7 Parameter Patterns Usage In Code

string_equals and *compartment_contains* are predicates applied on parameters to select the appropriate ones by name or stratification. *N_AGE_GROUPS * N_COMORBIDITY* checks if we are selecting 32 parameter values. *set* takes a single value and *set_array* takes an array of the same length as the number of selected parameters. An objective for the project is to eventually provide an alternative for users to parameterize their equations without code.

5.1.5 Model Validation

Model validation involves comparing the expected output of a model with the actual output. The output may be thought of as the graph produced for an entire simulation, but, in practice, this may change based on simulation software. The derivative produced given a population distribution, however, is predictable.

In order to compare models, randomly generated population distributions can be used to compare both models' derivatives. The reference model's derivative is the expected derivative of the model being validated. In the event where the two are identical, the validation is done. In the event where the two are different, we need to refine the incorrect model and ideally identify where the differences come from. This is an aspect of the project which presents an opportunity for usability improvements.

For the context of this thesis, a validation script was produced which iterates over the names of our flows of this specific model and produces derivatives for both models based on an input. The original R code was transpiled to Python and its equations were deconstructed in order to map the terms to names of equation. By writing a validation script which validates a projection of the model along a single equation, it is much easier to refine. Since flows in our platform are named and names are kept in the artifacts, we can easily toggle certain sets

of equations by name. In the future, model validation may be further automated so that little to no programming may be involved for similar use cases, rendering this task simpler for epidemiologists.

5.2 Evaluation Design

This section is dedicated to the identification of relevant evaluation practices. Performing an adequate evaluation will require the use of software quality standards and modeling tools evaluation guidelines. There exist procedures depicting processes and considerations for evaluating domain specific modeling tools, which we will attempt to use. The current objective is to build an evaluation process based on proposed practices which aligns with the high level concerns for our project.

An evaluation is required not just for the sake of this thesis, but also in order to identify flaws in our current practices for the future of the project. The platform we are developing presents fundamental differences in its design and capabilities when compared to other epidemiological modeling approaches. It is imperative to not only validate or invalidate the integrity of the project as it stands but also provide insights into what needs to be addressed in the future.

5.2.1 Domain Modeling Evaluation Practices

The most direct approach to produce an evaluation could be to follow MDE or domain specific modeling evaluation practices. It may be possible to reuse tests or criteria commonly used in similar evaluation contexts.

Ontology Based Approaches

Fundamental approaches attempt to evaluate metamodel usability based on how closely they are able to model real systems. These systems may be fully described through ontologies. Evaluating a metamodel using ontology based techniques requires a reference ontology encapsulating a deep understanding of the field [39]. Given that we have not been able to identify previous work on describing epidemiological models in terms of model driven engineering, we do not have a reference ontology for compartmental epidemiological models.

There exist ontologies describing diseases in epidemiological terms [40] and describing artifacts used in epidemiological modeling [41], but their degree of usefulness regarding meta-modeling is limited. Most of the concerns which can be extracted from the ontologies show their usefulness during model parameterization, and some during model design. Some exam-

ples are disease transmission dynamics or relationships with comorbidities. This information is useful for users attempting to ensure that all the information known about a disease is integrated in their model. It is unclear how the structure which they have chosen to represent information, the ontological meta-information if you will, can be of use to guide the design of the metamodel.

Instead of a reference ontology, we refer to the description of the field by an epidemiologist and transfer his view of the project in terms of MDE components. Description of these components and their relationships, which we directly translated into a metamodel, closely resemble an ontology. Following the proposed approach, a fair evaluation of our metamodel would require a published and accepted ontology describing all kinds of compartmental epidemiological models.

Research suggests evaluating domain and comprehensibility appropriateness using a domain ontology [39]. The authors denote properties of languages, dialogue, information and modeling which establish a correspondence between the domain modeling software and the understanding and representation of the domain. They propose four properties a modeling language should respect: lucidity, soundness, laconicity and completeness. The properties refer to the cardinality of relationships between language elements and domain concepts. An ideal language has a one to one correspondence between its elements and concepts of the domain.

A parallel can be made between the relationship of a metamodel to a domain and the relationship of a model to a metamodel. Following this point of view, it is possible to view models for which the elements closely match elements of the metamodel as preferable. Our situation follows this guideline very closely, as our metamodel is quite simple and each class it defines can be directly instantiated in a model.

While there is value to be found in using ontological approaches for metamodeling of physical system, this may not transfer to epidemiology. The proposed approaches for evaluating domain appropriateness and modeling language completeness seem correct, but do not directly translate to functional appropriateness. Specifically, ISO 25010 describes functional appropriateness as the "degree to which the functions facilitate the accomplishment of specified tasks and objectives." [42] For the domain appropriateness of a modeling language to directly translate into software functional appropriateness, we have to demonstrate that what epidemiologists want to model with our tool is exactly what would be described at the domain ontology.

Our tool can act more like a framework than a physical system modeling tool. By providing adequate composition and abstraction during modeling we allow for operations which do

not necessarily represent epidemiology concepts but allow for the creation of epidemiological models nonetheless. Our problem may be to *produce* graphs and equations but that is not what we actually want to *model*. Take for example the Cartesian product operation, an incredibly useful feature for epidemiology. Even after being identified as such, it seems to only appear as a top level operation in most platforms or models, it is never used as a composability tool. The model presented in the case study presented a perfect opportunity for a Cartesian product right in the middle of the structure but the authors did not present it or implement it that way. This simply means that one could define an ontology for compartmental epidemiological models and omit the fact that Cartesian products can be treated like other elements. According to the ontology based evaluation principles, a platform which introduces Cartesian products as a model element could be imperfect. I do not believe it is possible to extract meaningful ontologies for epidemiological models by making use of existing models which are built and presented using limiting representations. The evaluation will not make use of an ontological approach.

General Approaches

A presentation from Sintef researchers [43] synthesises existing evaluation processes for domain modeling software and connects it back to the purpose for creating these tools. For example, the authors note a challenge: "It is often far from evident that a DSL might be useful or that developing one might be worthwhile" [43, p.6]. They finish by identifying product characteristics such as "easy to learn, problem appropriateness, effort-saving" [43, p.26]. A positive evaluation for the identified characteristics is a valid argument for the creation of a DSL. While the authors note that little work has been conducted on how to achieve desired quality characteristics [43, p.10], they offer a range of qualities which we can evaluate at the current state of our project. Notable characteristics [43, p.14] are essentially what the ISO/IEC 25010 standard [42] describes as functional suitability and maintainability.

5.2.2 Software Quality Evaluation

There seems to be little difference in terms of process for evaluating these characteristics for domain specific modeling software compared to other types of software. We will cover general software evaluation standards, specifically ISO/IEC 25040: Evaluation process [44] and ISO/IEC 25010: System and software quality models [42].

ISO/IEC 25040 suggests identifying a purpose for the evaluation. Our evaluation's purpose is to validate or invalidate the work and practices applied for this thesis in order to provide feedback to current and future members of the team. Any aspect of the project is within

the evaluation's reach but it mostly focuses on the validity of the metamodel and resulting models. Aspects such as software architecture and implementation will be mostly ignored to align with the main objectives of the thesis, applying MDE to epidemiology.

It also suggests gathering requirements from stakeholders. We have presented user stories in Table 4.1, essentially requirements from users' perspectives described in terms of tasks and features. From these, we can build more specific criteria for our evaluation.

ISO/IEC 25040 recommends describing evaluation measures in terms of software qualities. This will allow us to, for example, identify a difficult modeling task and qualify it as a modifiability issue, a software quality.

ISO/IEC 25010 classifies software qualities, of which two classes are of interest to us. Both at the platform level and at the model level, we wish to evaluate functional suitability and maintainability. Functional suitability corresponds to completeness, correctness and appropriateness [42]. Maintainability includes modularity and modifiability, as well as other desired qualities not within the scope of this thesis, such as analysability and testability [42].

The following qualities are of interest for evaluation:

1. Platform correctness
2. Platform completeness
3. Platform appropriateness
4. Model modifiability
5. Model modularity

Above, platform refers to the software we are developing, mainly the metamodel, while model refers to epidemiological models created using our platform. The following describes evaluation guidelines for each of the identified qualities in order to address the epidemiological modeling issues we aim to resolve.

Platform Correctness

Correctness is the "degree to which a product or system provides the correct results with the needed degree of precision." [42] Our models are deterministic and always produce the same physical representation and equations, so precision is not a problem in that sense. For equations not supported natively, other equations or compartments may be implemented by a metamodeler to solve specific problems and this points to platform completeness rather than

correctness. Helping the users avoid models which produce unexpected results is related to correctness but does not correspond to any form of degree of precision. Some of the concerns that may be attributed to correctness are covered by completeness and appropriateness.

In order to evaluate this quality, demonstrating the ability to reproduce expected results from generated mathematical equations will be enough. This validates that our metamodel code generates appropriate physical representations for models and that our simulation software correctly evaluates mathematical expressions.

Platform Completeness

Completeness is the "degree to which the set of functions covers all the specified tasks and user objectives." [42] As our group learns about epidemiology, epidemiological modeling seems more and more like an open set of problems. As of now, every identified problem has a solution given our implementation. Typical modeling problems are covered on purpose, while atypical problems are covered because they can be expressed using the same components.

Evaluating completeness quantitatively is not possible, we will replace this with a qualitative evaluation. The discussion will cover the composability model developed for compartments and flows, responsible for allowing emergent modeling behaviour.

Platform Appropriateness

Appropriateness is the "degree to which the functions facilitate the accomplishment of specified tasks and objectives." [42] Epidemiological modeling tasks we aim to facilitate revolve mostly around model modification, especially doing so safely. For example, one of the objectives of our platform is to avoid typical copy paste errors when adding components to a model. Platform appropriateness in this sense corresponds closely to model modifiability and modularity, qualities we also evaluate. We will reflect on the case study and extend the discussion to example model modifications and evaluate the safety and ease of identified tasks. Though safety is not mentioned in the software quality description, it has to be included as the idea is to produce correct models, not just any model, with the tools we provide.

Model Modifiability and Modularity

Modifiability relies on modularity, so they are presented together. Modifiability is the "Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality." [42] Modularity is the "degree to which a

system or computer program is composed of discrete components such that a change to one component has minimal impact on other components." [42]

Our particular epidemiological metamodeling approach is a study in platform appropriateness, it allows us to model components such as flows or compartments in a way previously unavailable. We have observed that our modeling approach enables more easily modifiable models by increasing modularity, which would demonstrate platform or metamodel appropriateness.

Modifiability and modularity evaluation will cover some elements of the model, while other similar elements will be ignored to reduce redundancy. Elements will be evaluated for modularity, assessing whether or not the element helps modularity, typically doing so by abstracting internal complexity or details from the rest of the model. Elements will also be evaluated for modifiability using the following modifications:

1. Extension:

- (a) Unit Compartment $\Rightarrow$ change to Group or Product
- (b) Group $\Rightarrow$ change to Product
- (c) Group $\Rightarrow$ Add compartment
- (d) Group $\Rightarrow$ Add flow
- (e) Product $\Rightarrow$ Add dimension

2. Simplification:

- (a) Group $\Rightarrow$ Remove compartment
- (b) Group $\Rightarrow$ Remove flow
- (c) Product $\Rightarrow$ Remove dimension

5.3 Platform Correctness Evaluation

As we have shown in the case study, our flows support parameterization patterns, allowing flexible but structured generated equations. The case study made use of most parameterization patterns, from which we were able to produce the same derivatives as the studied Ontario COVID-19 model. The verification was done using the original source code from the GitHub repository¹ using arbitrary population distributions (distributions over risk, age, infection, etc). All observations indicate that the platform generates correct equations.

¹github.com/ashleighrt/Ontario-COVID19-model

Verifying Numerical Results The original implementation produces random distributions of infected individuals, that is isolated or not isolated as well as having comorbidity or no comorbidity. This can be problematic when attempting to reproduce the model, as when the output is different, it is nearly impossible to diagnose. For this reason, the verification of the validity of the produced model relies on fixed distributions of population. The following presents numerical results given a single distribution generated from the original R code, but results have been tested using other distributions.

The distribution only presents value of susceptible people, mildly symptomatic and severely symptomatic. Table 5.1 presents the number of individuals of each category as columns of 32 values for the 16 age groups and two comorbidity groups.

Table 5.1 Population Distribution

Age	Comorbidity	Susceptible	Mildly Symptomatic	Severely Symptomatic
0-4	no	650332	1	1
5-9	no	682299	1	0
10-14	no	703567	2	1
15-19	no	748657	4	1
20-24	no	888276	3	1
25-29	no	936968	6	2
30-34	no	877531	8	1
35-39	no	800775	8	1
40-44	no	736630	6	2
45-49	no	671913	10	3
50-54	no	691037	10	6
55-59	no	665423	13	7
60-64	no	477320	15	5
65-69	no	346498	19	2
70-74	no	242304	23	9
75+	no	363103	18	1
0-4	yes	77923	21	8
5-9	yes	81754	18	2
10-14	yes	84302	25	7
15-19	yes	112859	23	5
20-24	yes	147011	28	6
25-29	yes	113444	23	8
30-34	yes	132286	28	5
35-39	yes	165178	30	5
40-44	yes	178438	24	6
45-49	yes	269142	36	10
50-54	yes	300408	33	4
55-59	yes	402671	28	8
60-64	yes	460441	36	13
65-69	yes	432149	39	8
70-74	yes	402121	22	5
75+	yes	723787	39	7

The R code is accessible using this *Colab link*. It relies on input code and data from our online repository, specifically found on the *tuite – model* branch. It is a slightly modified version of the original R code which instead of producing a simulation output, produces only the derivative given a distribution. This derivative is a matrix of 512 compartments, the presented results cover only the first 32, the susceptible compartments, as the rest are available online if necessary.

The Python code is also accessible using the *tuite – model* branch of our public repository. This branch contains, in the *src/simulation* directory, a verification script, code to produce parameterization for our model and code corresponding to the translation from R to Python of the original model.

The results are nearly identical, except for numerical approximations present during the extraction of the original parameters. Table 5.2 presents the derivative of the 32 susceptible compartments, for age and comorbidity, given the original distribution. The total derivative is 512 compartments, which can be produced using the verification script in the repository. The numerical results are the sum of aging, exposure, infection, symptoms, hospitalization, recovery, recovery-h, enter-ICU, exit-ICU and death as presented in the original model. Note the first, the sixteenth, the seventeenth and the thirty-second rows contain large numbers due to aging. A large number of people leave the 0-4 age group while a large amount enters the 75+ age group. The differences in values between the original model and the platform output are essentially due to numerical approximations and seem negligible.

Table 5.2 Derivative of Susceptible Compartments

Age	Comorbidity	Original R	R Translated to Python	Platform Output
0-4	no	-363.077700073081	-363.07769637101666	-363.07769637101666
5-9	no	-11.1555903031153	-11.155569831457512	-11.155569831457504
10-14	no	-12.4705109853127	-12.470506148692834	-12.470506148692857
15-19	no	-14.051299419946	-14.051309656826334	-14.05130965682635
20-24	no	-13.4435315740779	-13.443555924991756	-13.443555924991768
25-29	no	-15.5693188190525	-15.569275490982704	-15.569275490982731
30-34	no	-13.2153303581084	-13.215362796177885	-13.215362796177908
35-39	no	-12.8544889121929	-12.854508480720597	-12.854508480720584
40-44	no	-12.1423873101538	-12.142380486985928	-12.14238048698592
45-49	no	-11.1961450691844	-11.196140160459947	-11.19614016045992
50-54	no	-10.1038864790398	-10.103871239951445	-10.103871239951458
55-59	no	-9.15229365555492	-9.15233073805507	-9.152330738055065
60-64	no	-6.38143776864892	-6.381411821573181	-6.381411821573174
65-69	no	-4.14913778909191	-4.149003032591224	-4.149003032591207
70-74	no	-2.52856758446857	-2.5287174446787617	-2.528717444678779
75+	no	353.528434557933	353.52844925071446	353.52844925071446
0-4	yes	-43.5040927138672	-43.50409227028461	-43.50409227028461
5-9	yes	-1.33667809807854	-1.3366750446067428	-1.3366750446067404
10-14	yes	-1.49422729759048	-1.4942348024665932	-1.494234802466593
15-19	yes	-2.11821381652169	-2.118214701747746	-2.118214701747748
20-24	yes	-2.22492448319752	-2.224921987133314	-2.224921987133317
25-29	yes	-1.88506523606846	-1.8850593790067478	-1.8850593790067505
30-34	yes	-1.99218397042695	-1.9921820079938115	-1.9921820079938144
35-39	yes	-2.65152979243633	-2.6515393066661517	-2.6515393066661486
40-44	yes	-2.94131831020895	-2.9413167264613875	-2.9413167264613884
45-49	yes	-4.48473667902008	-4.4847209921423925	-4.484720992142396
50-54	yes	-4.39236731086095	-4.39237578607721	-4.392375786077212
55-59	yes	-5.53837669959696	-5.538388490295965	-5.538388490295965
60-64	yes	-6.15577723044185	-6.155772175020189	-6.155772175020185
65-69	yes	-5.1747650676722	-5.174763988887555	-5.1747639888875545
70-74	yes	-4.19634065320459	-4.196334565110256	-4.196334565110258
75+	yes	37.0805733120567	37.080568603366515	37.08056860336652

5.4 Platform Completeness Evaluation

Platform completeness is not closely related to the modeling case study, as it refers to the ability for metamodelers to correctly implement composable compartments using our framework. During the case study, the births flow was left out in order to not introduce a third metamodel extension which is outside the desired scope of the evaluation. The *cross-rates* modeling extension, as described in the metamodel extension requirements section, can adequately represent births, so this is not considered a completeness issue.

A second difference is present in the equations from our model and the studied model. The difference is found in the contact equation. A division by the total alive population at a given time is required, but our model instead uses total alive population at time zero. The approximation is negligible as the studied model is not an apocalyptic scenario. Solving this problem is not trivial as there are many solutions and identifying the most appropriate one was outside the scope of the case study.

There is a relationship between these two changes, as using a trivial implementation of the division by the alive population is incompatible with a trivial implementation of *cross-rates* for births. This is because *cross-rates* for births may draw from a negative population compartment in order to create births. The modeler then has to specify that the compartment from which births originate should not be counted in the sum of alive populations. This is not so much a completeness issue rather than an appropriateness issue. There are many imaginable metamodel extensions for both division by alive population and for birth flows which help with different aspects of this task. Different extensions can be offered and users may form their judgement of which is most appropriate.

5.5 Platform Appropriateness and Model Modularity and Modifiability Evaluation

Given the tree structure of our epidemiological models, the evaluation is performed in topological order. The first evaluated elements are found in the stratified model displayed in figure 5.5. Once this is done, elements shown in figure 5.3 will be covered. Elements similar to those that were covered previously will be mentioned and point back to the element they are analogous to, reducing the redundancy.

5.5.1 Top Level Element

The top level element is a product between three dimensions: age, comorbidity and the disease model. As a product, it is subject to two tests: adding and removing a dimension.

Adding a dimension: Two plausible examples of such a task are in order to provide finer parameterization to the disease model, say the duration of the latent or infectious period or the likelihood of the disease being fatal. First, we could add a dimension to distinguish between male and female population if statistics indicate relevant differences. Second, we could decompose the comorbidity if we note a dependence between two of them or if one of them is more important. For example, it might be the case that asthma is the most important condition to determine the probability of entering ICU, if so, it would be wise to denote whether or not individuals have asthma separately from other comorbidities.

Both of these changes have identical impact in terms of parameterization and involve no further changes to the model. The impact on parameterization is necessary as the reason for such a change is to provide more accurate parameters based on our new dimension. The only downside is the appearance of this specialization in every equation, for example equations such as rate from pre-ICU to ICU may be unaffected, but now require additional values. This was a design decision, as using under-specified parameters is possible, but could lead to modelers simply forgetting to specify parameters after such a change.

In the sex example, there is no change in the general level of modularity as we introduce a new modeling aspect, however this is not the case for asthma. Introducing asthma as a new element reduces modularity as there are now two elements related to comorbidity placed next to each other arbitrarily. Ideally, our comorbidity model should be composed of both asthma and the other diseases. Doing so would also open the discussion to improving such a comorbidity component, perhaps introducing a product with a dimension for each disease, perhaps introducing rates from *None* to *Some* for each disease. The ability to group similar concerns in a component which can stay opaque to the rest of the model definitely respects our modularity concerns. Changing the layout of the elements while not changing the total number of physical compartments or equations generated typically leads to no required change in the parameterization code, which respects our modifiability concerns.

We conclude that adding a dimension to a product will rarely be an atomic change, as it introduces new modeling concerns. Consequential required changes are not a downside, as they are either the reason for the change originally, such as added parameterization capabilities. In the event where newly modeled concerns coincide with existing ones, it allows the user to reason about their change and potentially group concerns into a single component which

allows easier future changes. Overall, this modification task is a positive for modifiability and modularity, the platform design makes modifications safe by requiring a given change in the model to be reflected in the parameters, otherwise producing an error.

Removing a dimension: This task is much simpler than addition. For this model, it may be of interest to remove age stratification or comorbidities. This would most likely lead to a worse model for predictions, but it can be interesting to see if various modeling concerns actually help the model perform better predictions. For smaller parts of the model, not top level stratification, such changes may be more relevant. This change specifically simply requires deleting the element and merging the parameters.

Much like other structural modifications, none of the unrelated equations have to be changed. This is in line with modularity as describe by ISO. Dissociating equations from structure is a design decision for our platform which seems to be appropriate.

5.5.2 Comorbidity

As a group, comorbidity is subject to five modification tasks, but removing a flow and removing a compartment are not applicable.

Change to Product: This task has been partially covered during the top level element evaluation. We previously described the separation of asthma and other comorbidities, which is a perfect example for this task. A most likely modification for comorbidity is to change it in a product of finer granularity diseases such as individual diseases for high impact diseases.

Adding a Compartment: This task is redundant as most uses are more easily covered by changing to a product.

Adding a Flow: This task is quite interesting as the data suggests that higher age groups suffer from a higher comorbidity rate. Intuitively, there should be a flow for developing comorbidity, as the model includes aging. For unknown reasons, this was not the case in the studied model, but very well could be included. This change is quite simple, as adding a rate from *None* to *Some* with a source parameter is sufficient. This allows us to parameterize this flow based on age to reflect the natural progression. It also allows us to increase the rate for infected or intensive care unit patients as we know COVID leaves long lasting symptoms in many cases.

None & Some Unit Compartments: These two compartments are omitted, as there is no logical extension to *None* and extensions to *Some* have been covered previously.

5.5.3 Aging

Since the Aging compartment is an instance of a metamodel extension rather than a product, group or unit, we proceed differently with the evaluation. Aging behaves as a group of 16 age compartments linked with 15 rates. In order to reuse the pattern of N compartments and $N - 1$ rates, the behaviour has been captured in a metamodel extension. Our platform promotes the use of such extensions, the following is why we believe this to be of functional appropriateness.

By making use of an abstraction over the age groups such as the Aging metamodel extension, we reduce technical debt assuming the component is reused across models. Development time is reduced by generating the compartments and flows, but users familiar with the extension also know by default that the first compartment is a source, the last is a sink, and it interfaces with births or other age groups. It also prevents faulty implementations such as representing mortality as the final age group, as the missing sinks would indicate a logical error, which plays directly into our modular component design, requiring an orthogonal component for an orthogonal concern. In terms of modularity, modifiability and appropriateness, this component is exemplary.

5.5.4 Disease Model Link

The final dimension of the top level product is a link to the disease model. Components in aggregates are typically values, but the link provides a reference to an external element as a value, it is a simple metamodel extension. Since the link is an instance of a metamodel extension, we proceed differently with the evaluation.

The ability to reference externally defined elements using links enables model level modularity. During the modeling process, modularity may be generally thought of as respecting the single responsibility principle for each component. What links offer is the ability to respect the single responsibility principle at the model level.

A first model replicates the researcher's model, shown in figure 5.1, while a second integrates it with a link. This practice may enhance reusability of models as there are some Canada specific stratifications such as our age groups granularity embedded in the model. This region specific knowledge does not have to be part of the disease model, so keeping it separate allows other epidemiologists to reuse only the disease model.

The use of a link also allows for modification of the original model to be reflected in all references to it, which could be of value or not. This is similar to updating code in a library referenced by clients, it has to be done with care. Using links also allows for easy replacement

of the referenced component with a different disease model in this case.

Similarly to the aging component, the link component respects modularity, modifiability and is an example of our platform appropriateness for epidemiological modeling tasks.

5.5.5 Disease Model

The disease model, seen in figure 5.3, is a complex group. It is made up of a product, a group, three unit compartments, four rates and a contact. It is subject to five modification tasks. We omit changing this group to a product, as it is of little value. This group is a dimension of a product, which means changing one of its dimensions to a product is equivalent to adding dimensions, which was already covered. We also omit any future addition or removal of flows, as they are redundant and essentially never impact future modifiability unless they break usual modeling practices.

Adding a Compartment: This task could be done in order to produce a closer match with the studied model. The studied model included quarantine, done using a compartment referred to as V , which could be in hopes of using it for vaccination as well. This addition was not done for the scope of this work as the parameterization was hard to understand. At the structural level, this change is very simple. Simply add a unit compartment V and two rates to and from S . Of course, much like seen before, no further changes are required at the model level, but parameters of affected equations need account for this addition.

This change is not ideal, however, from a philosophical point of view. Certainly it is exactly as it is implemented in the reference model, but it is not how we recommend this change to be performed. Specifically, this implementation is too restrictive, only allowing for susceptible people to be under quarantine. The ideal approach would be to represent quarantine as a dimension of the top level element. It would allow for the same result as the researchers' implementation, and more. For example, it allows modeling quarantine for symptomatic people. The researchers' approach is not necessarily wrong, especially if it allows to model exactly what they intend to model, but it can be misleading.

Our first step against this class of logical error, much like in software engineering, is the use of correct names. The component we are presently evaluating should be referred to in the model as the "disease model". Seeing components such as hospitalization, isolation and quarantine in this model would be a red flag, as they are not inherently linked to the disease, simply correlated, much like aging and comorbidity. For this, our platform may be of superior appropriateness to nearly all other tools, as the use of names for each compartment and flow helps organize components coherently. Doing so typically increases modularity and reduces

the impact area for typical modifications.

Removing a Compartment: Following the advice given in the previous section, this change will focus on the removal of the *Dead* compartment. As noted previously, death, much like hospitalization or isolation, is correlated with the disease but not part of the disease model. This iteration of the model does not even allow people that do not reach the icu to die, which is unusual as there are plenty of reasons for death other than COVID. Of course, the argument can be made that in the scope of this model it is not relevant, but as we know, COVID is most likely not going away completely. COVID models will have to be adapted to support long term modeling like seasonal flu. What we recommend is simply moving mortality out of the disease model and into the top level compartment as a dimension, much like the previous recommendation for quarantine.

A pattern can be seen emerging where a compartment is drawn out of a group and placed into the above product as a binary option. This pattern could be implemented as a model transformation in the future for our platform and provide easier support for our vision. There are downsides to representing concerns as dimensions of a product, flexibility is not free. It requires writing and reading more parameterization code and may obfuscate intent when looking at the model. If our platform achieves adoption, time will tell if epidemiologists prefer this approach or not.

The idea that concerns such as aging, comorbidity, isolation, quarantine, mortality, and certainly others and be grafted onto the top level product is quite liberating. It presents optimal modularity and a path to add any eventual concerns onto existing models. Most if not all concerns, such as vaccination, geographical area, working remotely, etc. follow this rule. Modifications such as our recommended approach for mortality which may simplify complex components such as the disease model are indeed of great value in terms of modularity and modifiability. The idea that this approach is enabled by product compartments, a key idea of our platform, is also valuable feedback for our metamodel appropriateness.

The remaining compartments of the model are comprised mostly of similar elements to the ones previously covered. For example, modifications regarding isolation, hospitalization and death have already been covered, while S and R offer limited modification opportunity.

5.5.6 Concluding Modularity

As theorized, our models are able to encapsulate epidemiological modeling concerns in components, respecting our modularity concerns. Typical modification procedures that were considered favored modularity as the natural method for representing concerns. Respecting

modularity seemed to be simple by design, but may not be intuitive for epidemiologists, as seen with the proposed changes in our platform for the studied model.

5.5.7 Concluding Modifiability

Components respecting modularity were generally seen as simpler to modify. Building models from objectively correct small components seemed preferable due to our modeling approach, which naturally favors modifiability. Grouping of elements and the use of products accelerates modifications by reducing duplication. Making use of composability tools such as groups and products is a direct gain in modularity by agglomerating related elements in a named element. Naming elements seems to provide intuition to the reader and helps identify logical errors in the process, such as two orthogonal concerns being wrongfully intertwined. The final positive for modifiability is that our models are only represented graphically as opposed to classical epidemiology where the graphical representation is an afterthought. This means that to validate classic models one has to look through the implementation, but our platform ensures that what you see is what you get.

5.5.8 Concluding Appropriateness

As noted previously, composability tools improve modularity and modifiability, which is a good sign as they are the basis for modeling using our platform. The tools we offer for modeling seem to compose as expected. The strength of composability tools is their behaviour regarding equations which operate on them, which we have shown to be correct in the context of typical epidemiological modeling tasks. The extension mechanism at the metamodel level helped accelerate development of the case study model with a simple aging compartment extension which is objectively correct, composable and reusable. The tools we offer at the metamodeling and the modeling level are undoubtedly of value, but time will tell if the proposed gains outweigh unfamiliarity for the epidemiological modeling community.

5.6 Evaluation Results

This conclusion to the evaluation compares our case study results with the overarching objectives of the thesis, being the improvement of:

1. Epidemiological model construction
2. Epidemiological model sharing

3. Epidemiological model reproducibility

Our hypothesis was that this could be done by applying MDE to epidemiology. This section validates the validity of the produced prototype and evaluation with regards to these three criteria.

5.6.1 Epidemiological Model Construction

We define this criteria as the ability to correctly implement models and assess their behaviour. We theorized that making use of an MDE approach would allow for a powerful graphical modeling tool capable of supporting any kind of epidemiological model. Correctness of the implementation has been confirmed by our case study results, while completeness seems sufficient, indicating the possibility to facilitate model construction. Facilitating construction can be expressed as a set of related concerns such as expressiveness, simplicity and flexibility. Some of these characteristics were captured during our case study.

Increased expressiveness was shown through the use of adequate names for compartments, composed aggregates and flows. In theory, this can also be done for classical epidemiological models but is not directly connected to the implementation. The ability to provide a graphical tool with which epidemiologists may adequately represent arbitrary composable epidemiological constructs is a step in the right direction.

Increased simplicity is mostly granted by the use of a graphical modeling tool. It is assumed that a graphical view of a model is closer to our understanding of this model than a programmed implementation might be. Reducing the amount of code makes modeling much less error prone, and avoids any mismatch between an implementation and graphical notation, as seen in the studied model during the case study. The only code present in our models is for parameterization and the future of the project includes plans for an integrated parameterization tool instead of Python API. No-code modeling combined with low-code parameterization greatly reduced the area in which errors may hide. Our approach to graphical modeling of epidemiological models seems simple, though it may be unfamiliar to epidemiologists using disease specific platform or programming their models.

Increased flexibility is arguable. One of the main concerns for our platform is to support the modeling of any disease, even combining diseases or modeling human population along with other species. There already exist platforms capable of facilitating the modeling of specific diseases in specific countries, which may be more appropriate than our generic platform. On the other hand, epidemiologists can program arbitrary models using general programming languages and see no limit to the flexibility of their implementations. Our platform lies

between these two extremes.

While we cannot compete in terms of efficiency with disease specific platforms nor in flexibility with programming languages, it seems that our platform is a very good compromise for both criteria. Competing with disease specific platforms in terms of appropriateness is possible given the ability for our platform to support such models. Competing with programming approaches is possible assuming complex constructs can be adequately represented in our graphical models.

The case study was not adequate for measuring flexibility. The studied model lies within the scope for which our platform is explicitly designed to support, as most communicable disease models are very similar. This is positive in the sense that our platform seems adequate for all these diseases. It is not sufficient to indicate if our platform supports spread behaviours for waterborne diseases, for example.

5.6.2 Epidemiological Model Sharing

A great side effect to the reduction of code as well as the decoupling of model and parameterization is the increase in ease of sharing. This, paired with our unique modeling approach, enables groups of users who share metamodel extensions, for example, those available in our repository, to share model files directly and execute them without any extra steps. Increased ease of sharing is a consequence of our use of MDE allowing implementation to be part of the metamodel as opposed to the model, which means models are only data. Since each epidemiological model is made up of one or more modeling project and two parameterization files, it is very simple to guide users to share their models adequately.

These aspects were not studied in the case study, as model sharing happens over time and the scope of the challenges to tackle is not fully determined. Further evaluation of whether our modeling approach facilitates model sharing will inevitably require the cooperation of epidemiologists or students in an appropriately sized study.

5.6.3 Epidemiological Model Reproducibility

The literature review identified reproducibility as a kind of crisis in the current state of scientific publications of epidemiological models. This is partially related to model sharing, as of course the ability to share models may completely negate some needs for reproducibility, but is only part of the problem. The larger problem is the sharing of incomplete models, say missing equations or parameters, incomplete or inadequate code, etc. Our approach to this problem is mainly to reduce the model to its graphical form and reduce the use of

programming.

Similarly to model sharing, measuring reproducibility in the case study is not possible. However, some of these reproducibility issues were encountered during the case study, specifically when attempting to reproduce the model without access to code. The model code was not shared with the article, but the article seemed to contain the required information, which was of course misleading as this ended up not being the case. In order to avoid situations like these in our platform, all equations and compartments are present in the modeling view, but also the parameters and how they are used in the equations, as described through the different patterns in the case study. Epidemiologists attempting reproducing a model from our platform to another will have all the information required directly available if they have access to the model. Sharing misleading information using our platform is essentially not possible, unless users share the wrong model, which is out of the scope of our work.

Furthermore, artifact generation completely decouples our modeling environment from the simulation environment. Users can share simulation artifacts to execute in a Python notebook with users who do not have access to the modeling environment. This allows users to simply share four small text files, a link to an online Python simulation runtime, and anyone can verify the output of the model. Verifying simulation results is only a small part of the process of reproducing epidemiological models, but trivializing this step is important. It is the final step and typically what non epidemiologists involved in the process might care for, as it indicates mortality, total infections, etc.

CHAPTER 6 CONCLUSION

6.1 Summary of Works

This thesis presents motivation, design, and evaluation of a prototype for a model driven epidemiological modeling tool. The objective of the thesis is to produce an epidemiological platform which facilitates all compartmental epidemiological model construction, sharing and reproducibility. This tool supports metamodeling as well as modeling, to compete in flexibility with any existing tool. It offers high level constructs such as operations over aggregates of elements and a flexible composability model. This is the first tool of its kind capable of offering a high level graphical work environment for epidemiological modeling of complex epidemiological models. We have shown that the use of such a tool can help mitigate some of the current problems faced by epidemiologists, such as reproducibility issues of epidemiological models published in scientific journals. Through a case study, we have validated that functional correctness and completeness are met, and that appropriateness, modularity and modifiability are increased using our platform. The evaluation concludes positively that the objectives of the thesis are met.

6.2 Limitations

The proposed approach to epidemiological modeling solves entire classes of problems related to programming and presentation of such models, but presents new problems instead.

First, some constructs represented in a model's implementation may not be directly implementable in our platform. We have assumed these cases to be negligible since they can be approximated using constructs available in our platform, but our tool will never be as flexible as a programming language, it is a trade-off to consider.

Second, it is unclear how reasonable the use of an extensible metamodel is for our project. Extensibility allows for complex operations to be offered through plugins made by users and easily shared. The alternative is to offer a closed metamodel, limiting expressiveness and flexibility, or periodically extending the metamodel. A fixed metamodel also allows for many MDE tools to operate. This is a second trade-off to consider.

6.3 Future Research

As seen in the evaluation section, many software qualities were left untouched, as the prototype is not concerned with the entire scope of qualities which are relevant for a finished product. Most importantly, usability was not measured as the current version has little to no usability features, which are definitely required for an adequate tool. A close contender is interoperability. Future research may lead to automated tools for transferring models between existing platforms and our own in order to facilitate adoption for entire communities of users.

The current direction of the project is to allow for climate sensitive infectious disease (CSID) modeling, for example, diseases transmitted by insects. We have shown the ability to model insect populations in our platform similarly to how we model human population, doing so should help bridge the gap between epidemiology for CSID modeling and other diseases. Unifying disease specific modeling tools under the hood of our platform by allowing for very generic modeling tasks, even non epidemiological constructs, is a key idea for the future of the project.

REFERENCES

- [1] M. S. Jalali, C. DiGennaro, and D. Sridhar, “Transparency assessment of covid-19 models,” 10 2020. [Online]. Available: [https://www.thelancet.com/journals/langlo/article/PIIS2214-109X\(20\)30447-2/fulltext](https://www.thelancet.com/journals/langlo/article/PIIS2214-109X(20)30447-2/fulltext)
- [2] S. M. Shariful Islam and A. Khosravi, “The need for a prediction model assessment framework,” 02 2021. [Online]. Available: [https://www.thelancet.com/journals/langlo/article/PIIS2214-109X\(21\)00022-X/fulltext](https://www.thelancet.com/journals/langlo/article/PIIS2214-109X(21)00022-X/fulltext)
- [3] J. Sills *et al.*, “Call for transparency of covid-19 models,” *Science*, vol. 368, no. 6490, pp. 482–483, 2020. [Online]. Available: <https://www.science.org/doi/abs/10.1126/science.abb8637>
- [4] C. E. Walters, M. M. Meslé, and I. M. Halla, “Modelling the global spread of diseases: A review of current practice and capability,” 05 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1755436517301135>
- [5] USGAO, “Infectious disease modeling: Opportunities to improve coordination and ensure reproducibility,” 05 2020. [Online]. Available: <https://www.gao.gov/products/gao-20-372>
- [6] J. Jin *et al.*, “Transparency, reproducibility, and validation of covid-19 projection models,” 06 2020. [Online]. Available: <https://publichealth.jhu.edu/2020/transparency-reproducibility-and-validation-of-covid-19-projection-models>
- [7] F. Brauer, *Compartmental Models in Epidemiology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 19–79. [Online]. Available: https://doi.org/10.1007/978-3-540-78911-6_2
- [8] M. Tracy, M. Cerdá, and K. M. Keyes, “Agent-Based modeling in public health: Current applications and future directions,” *Annu Rev Public Health*, vol. 39, pp. 77–94, Jan. 2018.
- [9] S. Varghese and B. Babu, “An overview on graph products,” *International Journal of Science and Research Archive*, vol. 10, pp. 966–971, 10 2023.
- [10] P. J. White and M. C. Enright, “Mathematical models in infectious disease epidemiology,” in *Infectious Diseases*. Elsevier, 2010, pp. 70–75.

- [11] W. Hamer, *Epidemic Disease in England: The Evidence of Variability and of Persistency of Type*, ser. Milroy lectures. Bedford Press, 1906. [Online]. Available: <https://books.google.ca/books?id=kmiiGwAACAAJ>
- [12] C. T. Bauch, *The Role of Mathematical Models in Explaining Recurrent Outbreaks of Infectious Childhood Diseases*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 297–319. [Online]. Available: https://doi.org/10.1007/978-3-540-78911-6_11
- [13] Eclipse, “Eclipse modeling framework.” [Online]. Available: <https://www.eclipse.org/modeling/emf/>
- [14] S. S. Musa *et al.*, “Mathematical modeling of covid-19 epidemic with effect of awareness programs,” *Infectious Disease Modelling*, vol. 6, pp. 448–460, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2468042721000154>
- [15] S. Del Valle, J. Hyman, and N. Chitnis, “Mathematical models of contact patterns between age groups for predicting the spread of infectious diseases,” *Mathematical bio-sciences and engineering : MBE*, vol. 10, pp. 1475–1497, 11 2013.
- [16] S. Coogan and M. Arcak, “A compartmental model for traffic networks and its dynamical behavior,” *IEEE Transactions on Automatic Control*, vol. 60, no. 10, pp. 2698–2703, 2015.
- [17] H. Øverby, J. A. Audestad, and G. A. Szalkowski, “Compartmental market models in the digital economy—extension of the bass model to complex economic systems,” *Telecommunications Policy*, vol. 47, no. 1, p. 102441, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0308596122001434>
- [18] A. G. Brolund, “Compartmental models in social dynamics,” 2021. [Online]. Available: <http://uu.diva-portal.org/smash/get/diva2:1577103/FULLTEXT01.pdf>
- [19] M. Chalak *et al.*, “On the risk of extinction of a wild plant species through spillover of a biological control agent: Analysis of an ecosystem compartment model,” *Ecological Modelling*, vol. 221, no. 16, pp. 1934–1943, 2010. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0304380010002425>
- [20] O. O. Oluwagbemi *et al.*, “Anospex: A stochastic, spatially explicit model for studying anopheles metapopulation dynamics,” *PLOS ONE*, vol. 8, no. 7, pp. 1–8, 07 2013. [Online]. Available: <https://doi.org/10.1371/journal.pone.0068040>

- [21] D. J. Gubler *et al.*, “Climate variability and change in the united states: potential impacts on vector- and rodent-borne diseases,” *Environ Health Perspect*, vol. 109 Suppl 2, no. Suppl 2, pp. 223–233, May 2001.
- [22] C. Koenraadt *et al.*, “The effects of food and space on the occurrence of cannibalism and predation among larvae of anopheles gambiae s.l.” *Entomologia Experimentalis et Applicata*, vol. 112, no. 2, pp. 125–134, 2004. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.0013-8703.2004.00186.x>
- [23] P. Parham and E. Michael, “Modeling the effects of weather and climate change on malaria transmission,” *Environmental health perspectives*, vol. 118, pp. 620–6, 05 2010.
- [24] T. Hernandez-Boussard *et al.*, “MINIMAR (MINimum Information for Medical AI Reporting): Developing reporting standards for artificial intelligence in health care,” *Journal of the American Medical Informatics Association*, vol. 27, no. 12, pp. 2011–2015, 06 2020. [Online]. Available: <https://doi.org/10.1093/jamia/ocaa088>
- [25] M. S. Jalali *et al.*, “Evolution and Reproducibility of Simulation Modeling in Epidemiology and Health Policy Over Half a Century,” *Epidemiologic Reviews*, vol. 43, no. 1, pp. 166–175, 09 2021. [Online]. Available: <https://doi.org/10.1093/epirev/mxab006>
- [26] A. Rodrigues da Silva, “Model-driven engineering: A survey supported by the unified conceptual model,” *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1477842415000408>
- [27] M. Pol’la, A. Buccella, and A. Cechich, “Analysis of variability models: A systematic literature review,” vol. 20, no. 4, p. 1043–1077, aug 2021. [Online]. Available: <https://doi.org/10.1007/s10270-020-00839-w>
- [28] J. de Lara, E. Guerra, and P. Bottoni, “Modular language product lines: A graph transformation approach,” in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 334–344. [Online]. Available: <https://doi.org/10.1145/3550355.3552444>
- [29] A. Cimatti *et al.*, “A comprehensive framework for the analysis of automotive systems,” ser. MODELS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 379–389. [Online]. Available: <https://doi.org/10.1145/3550355.3552408>

- [30] H. S. Borum and C. Seidl, “Survey of established practices in the life cycle of domain-specific languages,” ser. MODELS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 266–277. [Online]. Available: <https://doi.org/10.1145/3550355.3552413>
- [31] R. d. A. Falbo, C. S. de Menezes, and A. R. C. da Rocha, “A systematic approach for building ontologies,” in *Progress in Artificial Intelligence — IBERAMIA 98*, H. Coelho, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 349–360.
- [32] “Composable c++.” [Online]. Available: https://github.com/CppCon/CppCon2021/blob/main/Presentations/Composable_C.pdf
- [33] J. V. Douglas *et al.*, “Stem: An open source tool for disease modeling,” *Health Security*, vol. 17, pp. 291 – 306, 2019.
- [34] A. Stewart Ibarra *et al.*, “Landscape mapping of software tools for climate-sensitive infectious disease modelling,” 02 2022.
- [35] M. A. BUI T. *et al.*, “The kendrick modelling platform: language abstractions and tools for epidemiology,” *BMC Bioinformatics*, vol. 20, no. 1, p. 312, Jun 2019. [Online]. Available: <https://doi.org/10.1186/s12859-019-2843-0>
- [36] J. P. PREVITE, “Graph substitutions,” *Ergodic Theory and Dynamical Systems*, vol. 18, no. 3, p. 661–685, 1998.
- [37] M. N. Woolfork *et al.*, “Empowerment and HIV risk behaviors in couples: Modeling the theory of gender and power in an african context,” *Womens Health Rep (New Rochelle)*, vol. 1, no. 1, pp. 89–101, Apr. 2020.
- [38] A. R. Tuite, D. N. Fisman, and A. L. Greer, “Mathematical modelling of covid-19 transmission and mitigation strategies in the population of ontario, canada,” *CMAJ*, vol. 192, no. 19, pp. E497–E505, 2020. [Online]. Available: <https://www.cmaj.ca/content/192/19/E497>
- [39] G. Guizzardi, *Ontology-Based Evaluation and Design of Visual Conceptual Modeling Languages*, 05 2013, pp. 317–347.
- [40] J. Beverley *et al.*, “Coordinating virus research: The virus infectious disease ontology,” *PLoS One*, vol. 19, no. 1, p. e0285093, Jan. 2024.
- [41] C. Pesquita *et al.*, “The epidemiology ontology: an ontology for the semantic annotation of epidemiological resources,” *J Biomed Semantics*, vol. 5, no. 1, p. 4, Jan. 2014.

- [42] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), “Iso/iec 25010: Systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models,” ISO/IEC Standard, ISO/IEC, International Standard 25010, 2011.
- [43] P. Mohagheghi and O. Haugen, “Evaluating domain-specific modelling solutions,” 11 2010, pp. 212–221.
- [44] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), “Iso/iec 25040: Systems and software engineering – systems and software quality requirements and evaluation (square) – evaluation process,” ISO/IEC Standard, ISO/IEC, International Standard 25040, 2011.