ASKE Phase 1 Report for AMIDOL

Eric Davis[1], Alec Theriault[1], and Ryan Wright[1]

[1]Galois, Inc

1 Introduction

Phase 1 of the AMIDOL project for ASKE has focused on proving out the initial concepts, theories, algorithms, and defining the core architectures, domain specific languages, and intermediate representations necessary to support the ambitious goal of advancing the state of the art of complex system analysis and machine-assisted science in ways that lower the overhead requirements in terms of large teams of domain experts, data scientists, mathematicians, and software engineers. AMIDOL: the Agile Metamodel Interface using Domain-specific Ontological Languages, aims to reduce the overhead associated with the model life cycle and to enable domain experts and scientists to more easily build, maintain, and reason over models in robust and highly performable ways, and to respond rapidly to emerging crises in an agile and impactful way. We do this by addressing the lack of generalizability, poor performance, and difficulties encountered when attempting to synthesize actionable knowledge and policies from the raw outputs of mathematical models.

AMIDOL is designed to support models in a number of scientific, physical, social, and hybrid domains by allowing domain experts to construct meta-models in a novel way, using visual domain specific ontological languages (VDSOLs). These VDSOLs utilize an underlying intermediate abstract representation to give formal meaning to the intuitive process diagrams scientists and domain experts normally create. AMIDOL then provides translations from these VDSOLs into an intermediate representation which can be transformed as appropriate to compose models, apply optimizations, and translate them into executable representations allowing AMIDOL's inference engine to execute prognostic queries on reward models and communicate results to domain experts. In Phase 2AMIDOL will bind these results to the original ontologies providing more explainability when compared to conventional methods.

AMIDOL addresses the problem of machine-assisted inference with two high-level goals:

1. improving the ability of domain experts to build and maintain models and
2. improving the explainability and agility of the results of machine-inference.

The VDSOLs help achieve these goals by lowering the barrier for entry associated with formal modeling languages. VDSOLs allow the expression of rich mathematical concepts using visual diagrams for systems and processes. The AMIDOL intermediate representation backs these diagrams with mathematical meaning, and allows the creation of new VDSOL elements as necessary with full expressivity, and Turing completeness. The use of the intermediate representation also allows AMIDOL to decouple the problem of model optimization, performance, and implementation from the task of model definition, ensuring any model defined in any VDSOL is translated to the same abstract representation. This abstraction can then be transformed using a set of well defined operations, eventually resulting in executable code which the AMIDOL inference engine can execute.

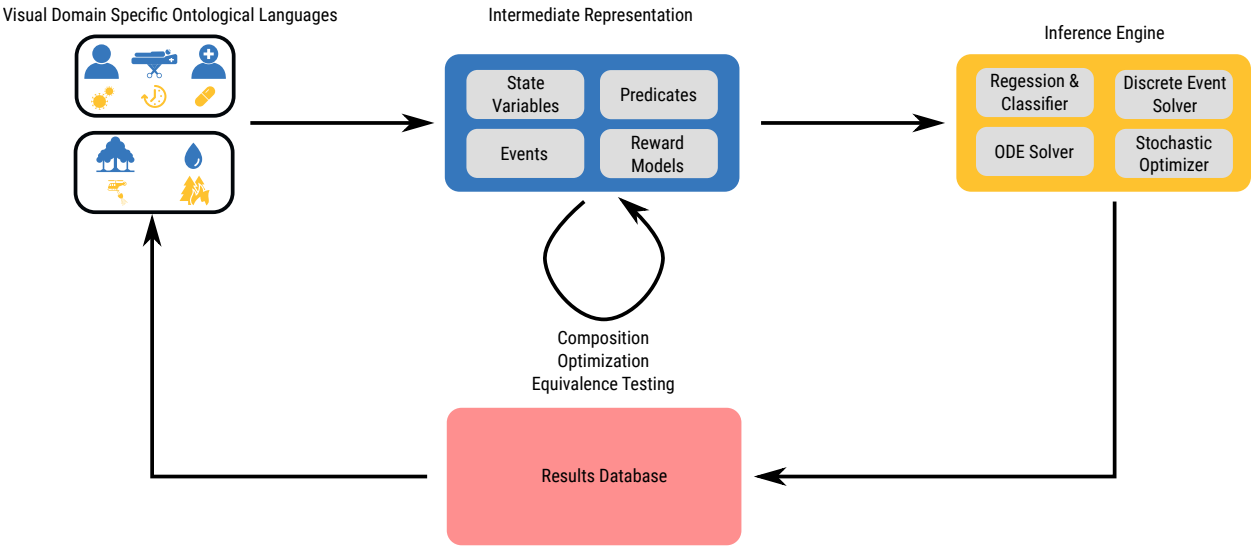


Figure 1: AMIDOL architecture.

Because AMIDOL uses a universal representation for this intermediate form, models defined with AMIDOL can be easily ported to off the shelf solution techniques which have already been vetted by the community, and optimized for high performance computing.

The Phase 1 Prototype for AMIDOL shows a proof of concept of the core capabilities required to achieve this full vision in Phase 2, and has helped to identify the necessary next steps in realizing a the full framework as part of the ASKE program. We include in our prototype three VDSOLs to show flexibility, and two backend targets already in use by the community.

2 AMIDOL Architecture

AMIDOL's architecture is illustrated in Figure 1. It consists of three main components, the UI Editor which allows for the definition of models in an appropriate VDSOL, the AMIDOL backend which translates to and from the AMIDOLintermediate representation, and the AMIDOL inference engine. Domain experts interact with AMIDOL by defining formal models using a visual domain specific ontological language (VDSOL) defined using a formal palette. These palettes consist of a set of nouns and verbs that can be connected via directed edges to form visual representations of the system in question. In Phase 2 we will extend this implementation to include a database of prior results, and a Design of Experiments interface which allows for complex analyses on models, and comparisons with past executions.

Once a model or models have been defined in a VDSOL, the AMIDOL backend translates the directed graph of nouns and verbs into the AMIDOL intermediate representation, or IR. The IR is a domain agnostic, universal, modeling formalism. The AMIDOL backend takes models represented in the IR and can perform model composition to build new models out of two or more existing models, can perform transformations on a model resulting in more performable versions of a model to allow for efficient solution of a model that takes into account structural optimizations, and the necessary requirements to solve a set of reward variables, and can be used to analyze two models to check for the equivalence of two models. The AMIDOL inference engine finally solves a model for a set of defined reward variables, and stores the results obtained in a results database.

The Phase 1 Prototype of AMIDOL implements initial versions of all of these components which are described in the remainder of this report, except the results database and Design of Experiments interface. During Phase 2 the features of these components will be expanded to full functionality, and the results database and Design of Experiments interface will be implemented.

3 AMIDOL VDSOL

AMIDOL is designed to support the definition of ontological languages which describe systems as formal objects. Objects for a given domain are organized into *palettes* consisting of **nouns** and **verbs**. Nouns define elements which make up the state space of a system, and verbs define transitions in the state space. VDSOLs enable domain experts to build models of complex systems which are easier to maintain, validate, and verify, and avoid common pitfalls of monolithic and hand-coded implementations. To provide visual context for modelers, AMIDOL supports the use of arbitrary scalable vector graphics (SVGs) to represent nouns and verbs, and features a canvas to draw nouns and verbs with labeled arcs connecting them to provide context. Each palette is defined using JavaScript Object Notation (JSON) using a schema which specifies the relevant properties of each known or verb, such as in the following palette for the SIR model:

Listing 1: JSON Definition of SIR VDSOL Palette

```
{"sir" : {
    "population": {
        "className": "population",
        "type": "noun",
        "classDef": "population.air",
        "icon": "images/person.svg",
        "inputVariables": ["P"],
        "outputVariables": ["P"],
        "parameters": [{"name": "P", "value": "0"}]
    },
    "patient" : {
        "className": "patient",
        "type": "noun",
        "classDef": "patient.air",
        "icon": "images/patient.svg",
        "inputVariables": ["P"],
        "outputVariables": ["P"],
        "parameters": [{"name": "P", "value": "0"}]
    },
    "infect": {
        "className": "infect",
        "type": "verb",
        "classDef": "infect.air",
        "icon": "images/virus.svg",
        "inputVariables": ["S"],
        "outputVariables": ["I"],
        "parameters": [
            {"name": "beta", "value": "0"},
            {"name": "total_pop", "value": "0"}
```

```
                ]
            },
            "cure": {
                "className": "cure",
                "type": "verb",
                "classDef": "cure.air",
                "icon": "images/cure.svg",
                "inputVariables": ["I"],
                "outputVariables": ["R"],
                "parameters": [{"name": "gamma", "value": "0"}]
            }
        }
```

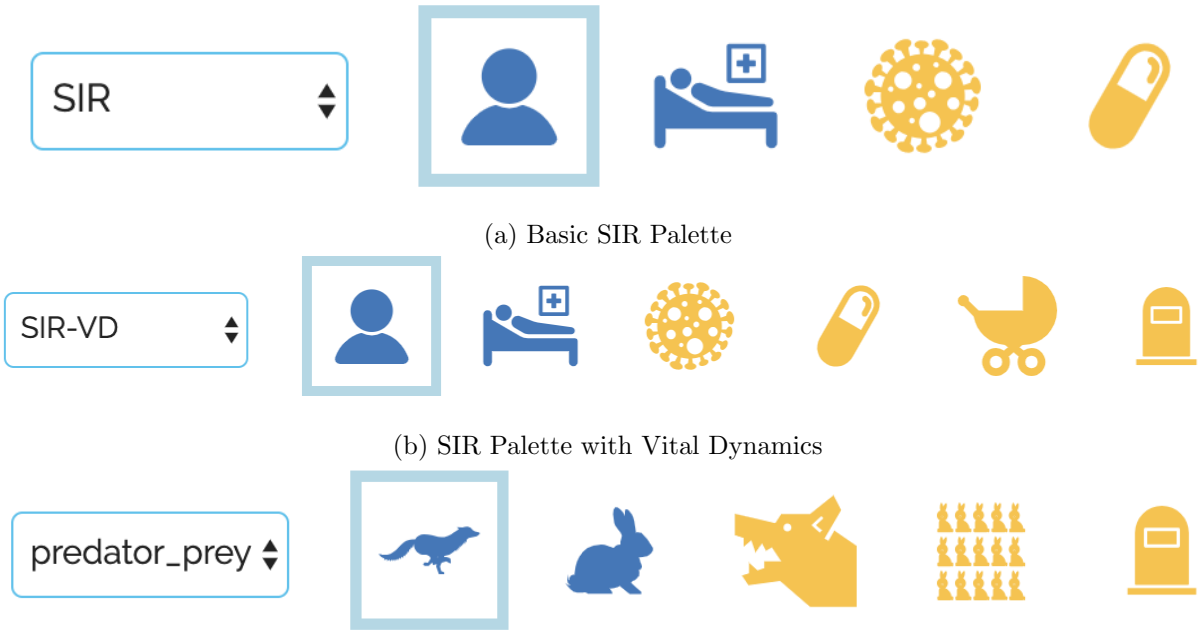
Each noun or verb is given a `class name`, a `type` (either noun or verb), a `class definition` as a pointer to a model in the AMIDOL intermediate representation, an `icon` to represent the element visually in the editor, and a set of `input variables`, `output variables`, and user specifiable `parameters`.

These definitions create VDSOL elements in an object oriented fashion, declaring nouns and verbs as abstract data types with public, private, and protected methods. The class as defined in the AMIDOL intermediate representation cannot by default be accessed elements outside of the noun or verb itself, and the `input variables`, `output variables`, and `parameters` are used to expose state variables or events as public or protected members for certain well defined interactions. The declared set of input variables expose state variables as protected members which can only be accessed by a VDSOL object of the opposite type, and with the same arity, i.e. nouns may only be accessed by verbs and vice versa and then only when cardinality of the input variables of the accessed object exactly match the cardinality of the output variables of the other object. In Phase 2 these conditions will be relaxed to allow multigraph compositions to provider richer modeling choices and capabilities. The set of parameters provide public interfaces which the user can specify as constants or expressions to further define interdependencies in a complex system.

The goal of AMIDOL's VDSOLs is to enable domain experts to define their models using an interface and visual language similar to the semi-formal diagrams they use today, but with the advantage that AMIDOLs VDSOLs have formal, executable, meaning. VDSOLs provide a performable, reusable, system for scientists to use when attempting to derive insights relating to the complex systems they represent. Figure 2 shows example palettes in AMIDOL which are implemented for the Phase 1 prototype.

AMIDOL is designed to be extended to support any domain through the use of palettes, and to allow researchers to draw diagrams using a visual language that they can modify, extend, and adapt to their needs. Because these palettes are backed by a universal intermediate representation, they are compatible and comparable even across palettes allowing models developed in differing VDSOLs to be contrasted, analyzed, and even composed.

In Phase 2 these capabilities will be explored with compartmental models for infectious diseases, composing models of populations with limited contact.



(a) Basic SIR Palette

(b) SIR Palette with Vital Dynamics

(c) Lotka–Volterra Predator-Prey Palette

Figure 2: Example AMIDOL Palettes

3.1 Structure of AMIDOL VDSOLs

Formally a VDSOL in AMIDOL is a directed bipartite graph consisting of nodes and edges, with nodes drawn from two classes, nouns and verbs.

Nouns : Nouns in AMIDOL represent portions of the model associated with its state space, though they may also contain events. Nouns are represented by custom SVGs, and can be connected to verbs which act upon them. Users can set the label associated with a noun, which impacts the naming of state variables and events associated with the noun, and set the value of any parameters exposed through the VDSOL parameter property allowing the nouns to be specialized with initial conditions, constants, or expressions.

Verbs : Verbs in AMIDOL represent changes in the state of the model, and which conceptually act upon nouns. Like nouns they contain state variables, unlike nouns they must also contain events. The state variables in a verb may be used to track state dependent rates, or internal properties, but also to define the side effects of the verb. A verb's input variables and output variables are used to connect verbs to nouns through composition.

3.2 Composability of Atomic Models

AMIDOL is being designed to support the composition of individual models in Phase 2 to enable model reuse, compositional methods for solution, to enhance backend support for performance optimizations that require symmetry detection, and to allow domain scientists to experiment by swapping out components of a model which may represent complex hypotheses about individual elements. Model composition in AMIDOL is being designed to support two primary mechanisms: *state-sharing* [25, 29] and *event-synchronization* [16].

Composition is currently partially implemented in the Phase 1 prototype, as it is employed as the method for building the IR model from the underlying definitions of nouns and verbs connected in the UI. In Phase 2 a model composition editor will be added to the UI to allow composing via state-sharing. State-sharing allows model composition by defining an interface for a model in the form of a set of state variables, which are then "shared" with another model. Shared state-variables in two composed models have the same marking, or value, and are effected by events in both models. Event-synchronization functions in an analogous way, allowing two events to be paired, such that the input predicate and output predicate of the new shared event is the union of the predicates of the events being shared.

AMIDOL will support replicate and join operations previously defined over Petri-nets and stochastic activity networks, but extended to the AMIDOL IR. [28] In addition to replicate and join operations, AMIDOL will support a novel model composition relation allowing users to specify sub-models as nouns and verbs which can be connected to existing components using Abstract Base Models, based on class inheritance patterns [4], which allows users to specify abstract models with undefined states and events.

4 AMIDOL Intermediate Representation

The Abstract Intermediate Representation (IR) for AMIDOL is meant to be a universal way to specify models, regardless of their domain, and provides a Turing-complete way to specify models performably, while avoiding domain specific considerations. Models are defined in the IR for AMIDOL using JSON for serialized representations using the following schema:

Listing 2: JSON Schema for the AMIDOL Intermediate Representation

```
{ "irModel": {
  "irModelName": "string",
  "stateVariables": [{
    "name": "string",
    "label": "string",
    "type": "sv_type",
    "initial_value": "expression"
  }],
  "events": [{
    "name": "string",
    "label": "string",
    "rate": "expression",
    "input_predicate": {
      "enabling_condition": "expression"
    },
    "output_predicate": {
      "transition_function": ["lvalue = expression", "lvalue =
         expression", ...]
    }
  }],
  "constants": [{
    "name": "string",
    "value": "extern"|"expression"
  }],
```

$$\text{translate}(\text{VDSOL}_A) \rightarrow \text{IR}_A \quad (1)$$
$$\text{compose}(\text{IR}_A, \text{IR}_B) \rightarrow \text{IR}_C \quad (2)$$
$$\text{translate}(\text{IR}_A) \rightarrow \text{target}_A \quad (3)$$
$$\text{optimize}(\text{IR}_A)) \rightarrow \text{IR}_B \quad (4)$$
$$\text{equivalence}(\text{IR}_A, \text{IR}_B) \rightarrow \text{True} \vee \text{False} \quad (5)$$

Figure 3

```
  "expressions": [{
    "name": "string",
    "value": "extern"|"expression"
  }],
  "rateRewards": [{
    "name": "string",
    "variable": "string",
    "temporaltype": "instantoftime"|"intervaloftime"|"
       timeaveragedintervaloftime"|"steadystate",
    "samplingpoints": [{
      "time": "float"
      }]
  }],
  "impulseRewards": [{
    "name": "string",
    "event": "string",
    "temporaltype": "instantoftime"|"intervaloftime"|"
       timeaveragedintervaloftime"|"steadystate",
    "samplingpoints": [{
      "time": "float"
      }]
  }],
  "composedrewards": [{
    "name": "string",
    "expression": "expression"
  }]
}}
```

The AMIDOL backend performs transformations on models defined in this IR allowing it to act on models via well structured transformations as shown in Figure 3. Even though we have not implemented the Model Composition editor for Phase 1, the backend composition operators have been implemented in our prototype as they form the basic operation needed to combine nouns, verbs, and reward variables into an executable model. The design principles used for the backend focus on this concept of reusable operations which enable complex interactions and features.

Currently the Phase 1 prototype implements only the `translate` and `compose` functionality shown in Figure 3, but in Phase 2 we will extend AMIDOL to account for all of this proposed functionality.

A core goal of AMIDOL is to use the intermediate representation to abstract away a lot of the details of target translations for solution, including necessary performance optimizations, conditions for solutions, and requirements of solution targets.

The intermediate representation employed by AMIDOL has its roots in Markov models [15], Generalized Stochastic Petri-nets with inhibitor arcs [6] (which have been shown to be Turing complete), and stochastic activity networks [19, 27] (an extension of Petri-nets that allow more compact model specification). AMIDOL currently extends these concepts primarily by creating ways to link to the original ontology of nouns and verbs, and by allowing embedded reward structures to be linked to a graph-based results database which stores the outcomes of experiments and can be used for the construction of arbitrary measures on the underlying model to support inference needs.

4.1 Language Properties

Formally, the IR is a 5-tuple, $(S, E, L, \Phi, \Lambda, \Delta)$ where:

- S is a finite set of state variables $\{s_0, s_1, \ldots, s_{n-1}\}$ that take on values in $\mathbb{N}$.
- E is a finite set of events $\{e_0, e_1, \ldots, e_{m-1}\}$ that may occur in the model.
- $L : S|E \rightarrow \mathbb{N}$ is the event and state variable labeling function that maps elements of S and E into the original ontology.
- $\Phi : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \rightarrow \{0, 1\}$ is the event enabling predicate.
- $\Lambda : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \rightarrow (0, \infty)$ is the transition rate specification.
- $\Delta : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \rightarrow N_0 \times N_1 \times \ldots \times N_{n-1}$ is the state variable transition function specification.

Informally the IR represents models defined in a given VDSOL using an formalism based on Generalized Stochastic Petri-nets with inhibitor arcs (which have the result of making Petri-nets Turing complete). Instead of inhibitor arcs, we utilize the more intuitive and performable method of allowing events to have input predicates (Phi) which can be evaluated to determine if an event is enabled, and output predicates which define the side effects of event firing.

State variables : intuitively, state-variables make up the current state of the model, and measure the configuration and capabilities of all modeled components. While state variables are defined as taking on values in $\mathbb{N}$, this does not restrict them from representing real numbers to arbitrary precision in modern computer hardware. In practice, they are implemented as integers, and floating point numbers by the AMIDOL source code.

Events : events, when fired, change the state of a model by altering the value of state variables. Events in AMIDOL are associated with input predicates, output predicates, and a rate function which returns the next firing time of a given event. The rate function is an expression over random variable distributions.

Input predicates : an input predicate is associated with an event, and a potentially empty set of state variables. Input predicates are functions of the marking of their set of variables which map the markings of those variables onto the set $\{1, 0\}$. For those markings in which the input

predicate evaluates to 1, the event is considered enabled and will fire as normal. For those markings in which the input predicate evaluates to 0, the event is considered disabled and cannot fire until subsequently enabled.

Output predicates : an output predicate is associated with an event, and a potentially empty set of state variables. Output predicates map a set of state variables, and their marking, to a new marking for the same state variables and define the side effects of event firing on the state of the model.

5 AMIDOL Reward Variables

The AMIDOL intermediate representation allows for the specification of reward variables or structures over a given model, and the composition of these structures with a model to produce composed models which can then be solved by the inference engine. Given a model $M = (S, E, L, \Phi, \Lambda, \Delta)$ we define two basic types of rewards structures, rewards over state variable values (rate rewards), and rewards over events (impulse rewards). [24, 8, 7, 26]

In Phase 1 we have only implemented reward variables, as a performable discrete event solver is required to fully validate an implementation of impulse reward variables (whose solution is otherwise trivial for numerical methods). During Phase 2 we will rapidly identify a discrete event solver as a backend target, and include impulse rewards as an early stage Phase 2 goal and deliverable.

5.1 Rate Reward Variables

A rate reward is formally defined as a function $\mathcal{R} : P(S, \mathbb{N}) \to \mathbb{R}$ where $q \in P(S, \mathbb{N})$ is the reward accumulated when for each $(s, n) \in q$ the marking of the state variable s is n. Informally a rate reward variable x accumulates a defined reward whenever a subset of the state variables take on prescribed values.

5.2 Impulse Reward Variables

An impulse reward is formally defined as a function $\mathcal{I} : E \to \mathbb{R}$ where $e \in E, (I)_e$ is the reward for the completion of e. Informally an impulse reward variable x accumulates a defined reward whenever the event e fires. Impulse rewards will be implemented in Phase 2, when the backend targets currently offered are expanded to include a performable discrete event simulator.

5.3 Temporal Characteristics of Reward Variables

Both rate and impulse reward variables measure the behavior of a model M with respect to time. As such, a reward variable θ is declared as either an instant-of-time variable, an interval-of-time variable, a time-averaged interval-of-time variable, or a steady state variable. An instant of time variable Θ_t is defined as:

$$\theta_t = \sum_{\nu \in P(S,\mathbb{N})} \mathcal{R}(\nu) \cdot \mathcal{I}_t^{\nu} + \sum_{e \in E} \mathcal{I}(e) \cdot I_t^e$$

Intuitively a rate reward declared as an instant-of-time variable [12] can be used to measure the value of a state variable precisely at time t, and an impulse reward declared as an instant-of-time variable can be used to measure whether a given event fired at precisely time t. While the latter is not a particularly useful measure (as the probability of an event with a firing time drawn from a

continuous distribution at time t is 0) it is defined for closure reasons, and for cases with discrete distributions and discrete time steps.

An interval-of-time variable intuitively accumulates reward over some fixed interval of time $[t, t+1]$. Given such a variable $\theta_{[t,t+1]}$ we formally define interval-of-time variables as:

$$\theta_{[t,t+1]} = \sum_{\nu \in P(S,\mathbb{N})} \mathcal{R}(\nu) \cdot \mathcal{J}^{\nu}_{[t,t+1]} + \sum_{e \in E} \mathcal{I}(e) N^{e}_{[t,t+1]}$$

where

- $J^{\nu}_{[t,t+1]}$ is a random variable which represents the total time the model spent in a marking such that for each $(s, n) \in \nu$, the state variable s has a value of n during the period $[t, t+1]$.
- $I^{e}_{t\to\infty}$ is a random variable which represents the number of times an event e has fired during the period $[t, t+1]$.

Time-averaged interval of time variables quantify accumulated reward over some interval of time. Such a variable $\theta'_{[t,t+1]}$ is defined formally as:

$$\theta'_{[t,t+1]} = \frac{\theta_{[t,t+1]}}{l}$$

Steady state reward variables are realized by testing for initial transients, and calculating an instant of time variable after a model has reached a stable steady state with high confidence.

For Phase 1 we have focused on instant of time variables. During Phase 2 we will implement interval of time and steady state reward variables. Steady state reward variables will make heavy use of our ability to perform transformations on the IR, as they represent an often wholly different solution target.

5.4 Composed Reward Variables

A key target of Phase 2 for AMIDOL will be extension of reward variables to allow composed rewards, which are expressions defined over other reward variables allowing us to construct a set of arbitrary measures on the set of atomic, and universal, measures of rate and impulse rewards. While individual reward variables form the basis for model evaluation, AMIDOL will support expressions defined over reward variables using basic arithmetic operations allowing reward variables to be composed via normal mathematical expressions.

6 Design of Experiments and Results Database

During Phase 2, a key extension of AMIDOL will be the implementation of a Design of Experiments interface [17], coupled with a results database in order to support the complex queries and prognostics, as shown in Figure 4, required of the program. Design of Experiments in AMIDOL will borrow from concepts of Plackett-Burman designs, and factorial designs using individual reward variables and parameterizations defined over a given model as atomic components of expressions. Computations for given atomic components will be stored in the results database to avoid unneeded reevaluation, and to allow future experiments to build on past results, and be directly compared and contrasted.

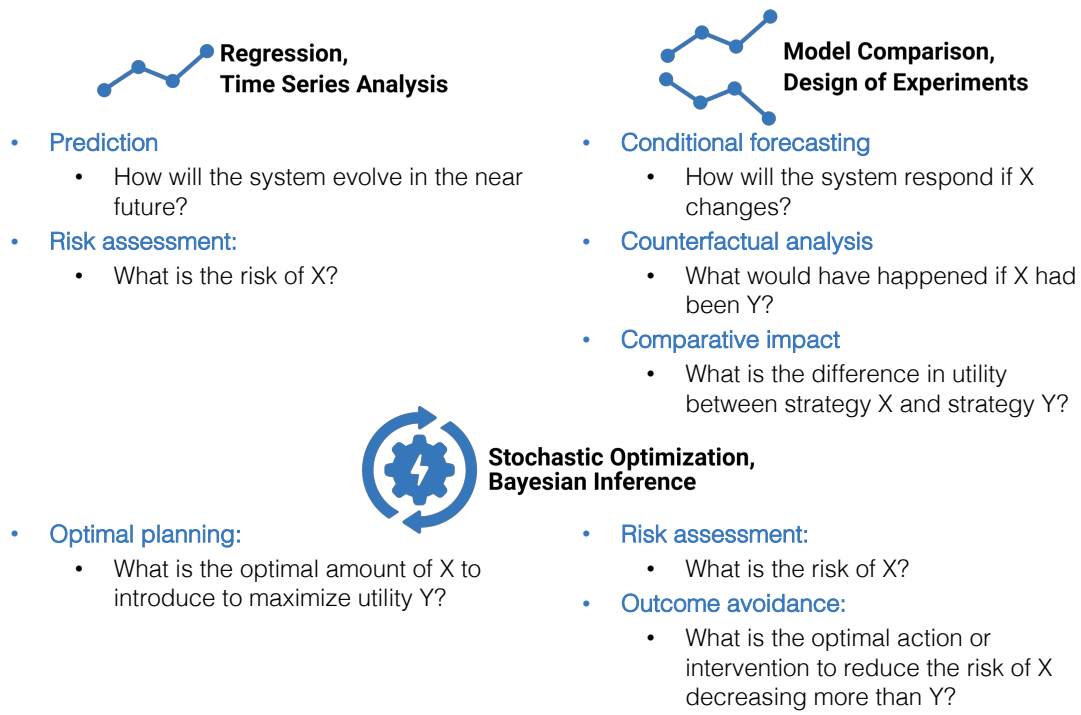


Figure 4

The Design of Experiments interface will also allow users to load data from external sources, and associate this data with expectations for state variables in the model. In our early tests we have achieved this by using data from the CDC's Fluview [5] data sets as series of instant-of-time observations. AMIDOL's interface will allow users to use regression to estimate the conditional expectation of dependent variables, given independent variables with fixed values from the data, allowing prediction, parameter estimation, and later risk assessment for a given regression through analysis of the distribution of possible realizations for state variables in a given model.

The planned Design of Experiments interface will allow direct comparison through user identification of reward variables, or expressions on reward variables, in two models which represent comparable properties or processes allowing the exploration of alternatives, conditional forecasting, counterfactual analysis, and comparative impact. Different strategies, configurations, or possible outcomes can be explored through examining different ways to parameterize a given model, or even differences in models with structural changes. Debugging experiments will be enabled through the connections to the VDSOL ontology. For instance, the interface will allow users to see the direct dependence of reward variables to noun's and verbs in the original ontology, better enabling researchers to understand the knowledge-based semantic dependencies normally hidden by traditional modeling techniques.

Sensitivity, uncertainty, and correctness measures can easily be constructed in the planned Design of Experiments interface. Because we allow for factorial and Plackett-Burman experiment design users can automatically perform one-factor-at-a-time [2, 20] sensitivity and uncertainty estimates. The initial prototype will also feature screening sampling-based methods [18] which have been shown to be computationally efficient and are further enabled by our VDSOL ontologies, as they help identify sources of uncertainty and error in the structure of the model. Correctness is enabled

by loading external data from multiple time series or sources, and asking the interface to perform cross validation on the input. AMIDOL will automatically support k-fold cross validation on time series, allowing automatic partitioning of data sets.

7 AMIDOL Inference Engine

The Phase 1 prototype of AMIDOL supports two solver targets for the intermediate representation PySCeS: the Python Simulator for Cellular Systems [22, 23], an extendable toolkit for analyzing compartmental systems, and through the use of the SciPy odeint functionality to simulate an ordinary differential equation based representation of a model in the IR [21, 1]. These targets were chosen both for reasons of validation, since they produce comparable output, but also as a proof of concept given their very different target languages.

The PySCeS solver has its own internal language which is used to specify compartmental models. AMIDOL supports this backend by providing a translation from the IR into the PySCeS specification language, and then invoking PySCeS on the translation. We illustrate a translation here by showing the outputs of AMIDOL for the basic SIR model in PySCeS:

```
Modelname: tmp_model
Description: Autogenerated tmp_model

Species_In_Conc: False
Output_In_Conc: False

# Reactions
reaction_4:
     Infected > Recovered
     0.333 * Infected

reaction_2:
     Susceptible > Infected
     0.413 * Susceptible * Infected /
        (Susceptible + Infected + Recovered)


# Parameter values
Recovered = 250.0
Susceptible = 250.0
Infected = 250.0
Example = 0.0
```

The SciPy odeint solver target is supported through a similar method by AMIDOL. A translation is provided from the IR into a Python script which includes and invokes SciPy's odeint solver for ordinary differential equations. The IR model is then represented directly as a system of ordinary differential equations for solution. We show here the same model previously generated for PySCeS as generated by AMIDOL for SciPy odeint:

```
from scipy.integrate import odeint
import json
import numpy as np
```

```
import matplotlib.pyplot as plt

class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# User defined constants
Example = 0.0

# The ODE system
def deriv_(y_, t_):
    Recovered, Susceptible, Infected = y_
    dInfected_ = 0.0 - 0.333 * Infected + 0.413 * Susceptible * Infected /
       (Susceptible + Infected + Recovered)
    dRecovered_ = 0.0 + 0.333 * Infected
    dSusceptible_ = 0.0 - 0.413 * Susceptible * Infected /
       (Susceptible + Infected + Recovered)
    return dRecovered_, dSusceptible_, dInfected_

# Boundary conditions and setup
timeRange_ = [ 0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0, 50.0,
55.0, 60.0, 65.0, 70.0, 75.0, 80.0, 85.0, 90.0, 95.0 ]
y0_ = 250.0, 250.0, 250.0
output = odeint(deriv_, y0_, timeRange_).T

print(json.dumps(output, cls=NumpyEncoder))
```

AMIDOL's backend inference engine is designed to support a wide range of inference techniques as build targets chosen by the intermediate representation based on the requirements of the reward variables being computed, and the limitations and requirements of the models being solved. A model, for instance, which has general event distributions, cannot be solved by successive matrix multiplication; the distribution of a reward variable can likewise not be computed by solving the underlying system of differential equations.

The strength of AMIDOL is this decoupling of modeling and representational concerns from the choice of executable implementation. AMIDOL removes these concerns from the domain expert, and automates these translations, allowing arbitrary modifications to be performed on the visual model, and rapid iteration of model structure, and parameters while exploring hypothesis, or comparing a model to actual data. While AMIDOL doesn't prevent a user from making mistakes while modeling, it reduces the occurance of implementation details, and helps users to more rapidly design and prototype models by removing abstract concerns, and details such as choice of language, or unfamiliar requirements of solution methods. While some work must be done to adapt AMIDOL to support a new solver target, once accomplished it can be leveraged for any VDSOL, allowing easy collaboration, and an ability to build on the work of others in a generalizable way.

A key feature for Phase 2 of AMIDOL will be the use of multiple solvers and techniques on the same model to compute different reward variables where appropriate, with the results being fed into a

results database. This backend architecture helps to ensure the performability of models designed in AMIDOL, and flexibility of solution technique helping researchers avoid the pitfall of being bound to a single target using hand coded models, which may or may not be performable.

8 Domain Models

We are currently testing AMIDOL using several domain models. We have selected a range of models to test different scenarios, use cases, and assumptions to aid in the prototype design of AMIDOL.

8.1 SIS/SIRS

The SIS/SIRS model is one of the simplest models we have deployed for testing with AMIDOL, with the advantage that the model itself is relatively simple, but utilizes real data, and can be used to answer important epidemiological questions. The primary objective of the SIS/SIRS model is to identify the *basic reproduction number* associated with an infection, also known as R_0, or *r nought*. R_0 was first used in 1952 when studying malaria and is a measure of the potential for an infection to spread through a population. If $R_0 < 1$, then the infection will die out in the long run. If $R_0 > 1$, then the infection will spread. The higher the value of R_0, the more difficult it is to control an epidemic.

The importance of estimating R_0 has been well established for many historical epidemics, including H1N1 [10] and Ebola [9]. This model also lends itself to testing AMIDOL's Design of Experiments module via the plentiful CDC Data [5] which is well modeled by SIRS.

Given a 100% effective vaccine, the proportion of the population that needs to be vaccinated is $1 - 1/R_0$, meaning that R_0 can be used to plan disease response. This assumes a homogenous population, and contains many other simplifying assumptions and does not generalize to more complex numbers. We have several main goals for SIS/SIRS models:

1. Fitting the models for the data in hindsight to perform goodness of fit estimates.
2. Finding the *retrospective* R_0 estimate over the entire epidemic curve.
3. Finding the *real-time* R_0 estimate while the epidemic is ongoing.

Data : For these models we have worked with the WHO/NREVSS (World Health Organization/National Respiratory and Enteric Virus Surveillance System) data sets at the resolution of Department of Human and Health Services designated regions.

Using data from a given region, and a given strain, we estimate R0 for the epidemic curve as shown in Figure 6

8.2 SIS/SIRS with Vital Dynamics

Extending the SIRS model, we also include an example of palette extension. The palettes defined for AMIDOL can be modified, and even extended, easily by adding new nouns and verbs. This allows for the long term curation of models by domain experts, who can easily extend their earlier work as new processes and interactions are discovered. The Vital Dynamics modification of SIRS allows the definition of models in which new individuals are born into populations, and for individuals to be removed from populations due to death from various causes. Increasesing the rate of infected

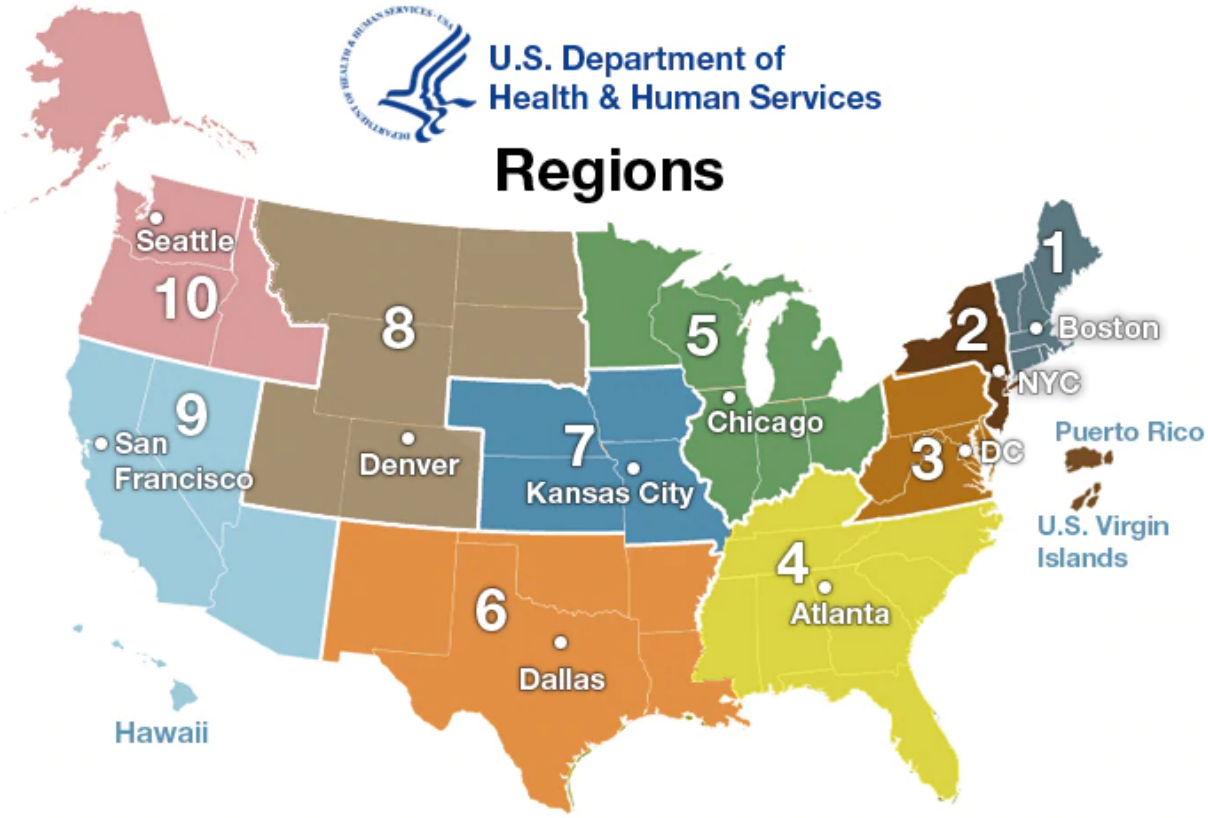


Figure 5: Department of Human and Health Services designated regions.

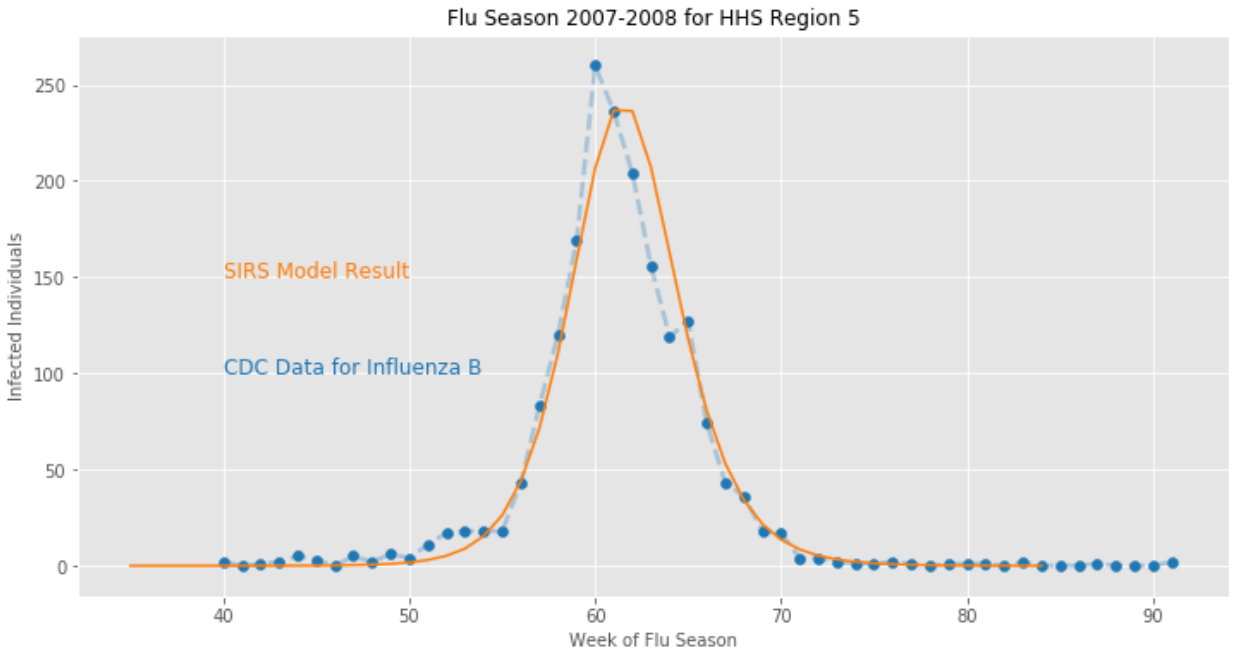


Figure 6: 2007 - 2008 Flu Season

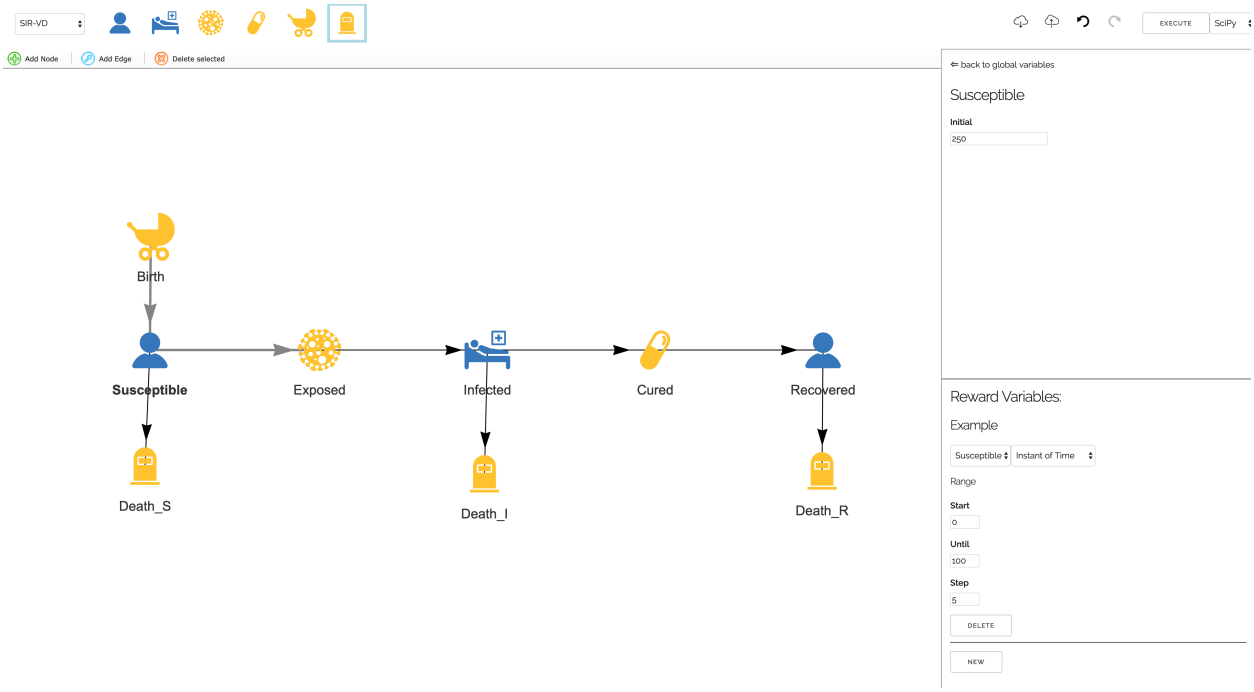


Figure 7: SIR Model with Vital Dynamics

populations dying, when compared to susceptible and recovered populations, also allows modeling a potentially lethal disease.

The following equations show the typical modification of the ordinary differential equations used to represent a SIR model with vital dynamics:

$$\frac{dS}{dt} = \mu N - \frac{\beta SI}{N} - \nu S \tag{6}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I - \nu I \tag{7}$$

$$\frac{dR}{dt} = \gamma I - \nu R \tag{8}$$

While these equations convey the underlying mathematical meaning of the model, their intuitive and semantic meaning is somewhat opaque. Figure 7 shows the implementation of this model in AMIDOL's SIR-VD palette. The result is a diagram that mirrors the sort of semi-formal diagram which has intuitive meaning to a domain expert and also has the same mathematical meaning as the system of equations due to AMIDOL's ability to perform a direct translation to the same underlying model.

8.3 Lotka-Volterra Model

As an example of a wholly different palette for a VDSOL, and to demonstrate the capability of our Phase 1 prototype to handle models in different domains, we have also implemented a domain model for the Lotka-Volterra model of predator-prey relationships [11, 3, 14]. Frequently used to describe the basic dynamics of biological systems with two species interactions, the Lotka-Volterra

model assumes a single predator, and single prey species. While a simple model, this system is an example of a Kolmogorov model which is used as the underlying framework for modeling other dynamic systems, such as competing species in a niche, the impact of disease on a species, and mutalistic relationships. It is provided in our initial prototype primarily as another model for which it is easy to validate and generate expectations, and to demonstrate the capacity of our prototype to work in multiple domains.

9 Code Repositories and Current Builds

The Phase 1 AMIDOL prototype is available, along with all source code in our repository with instructions on building and executing the current system. Models can be designed in the current VDSOL editor, a javascript front-end. The intermediate representation is implemented as a Scala project which compiles queries issued by users for a particular model they've created into code artifacts targeting different solver backends. Over time, we expect this system to also manipulate and compose datasets (from real-world observations as well as from the output of other simulations), and utilize the results database to save intermediate results and compare evaluations and models.

9.1 Front-End Architecture

AMIDOL's user interface is currently implemented using HTML, CSS, and Javascript, communicating with the Scala back-end using JSON over HTTP. This client/server approach leverages standard browser technologies for rapid development of rich interactions tailored for the specific needs of scientific modeling. It also decouples the concerns of visual presentation and manipulation from the underlying representations of model semantics and helps to hide the details from domain experts reducing their cognitive load when designing models.

The current user interface implements a rich visual editor allowing users to select a palette to work in, to save and load VDSOL diagrams, undo and redo operations, to customize parameters exposed by nouns and verbs, to define reward variables on a diagram, and to call multiple backend solver targets to solve those reward variables. Currently, the user interface implements a visual editor for directed graphs having labeled vertices and edges, which represent the nouns and verbs of a VDSOL.

Figure 8 shows the standard SIR model parameterized to fit real data from the CDC's Influenza B infections for Health and Human Services Region 5 during the 2007-2008 flu season. Executing this model gives us the results shown in Figure 9.

Figure 10 shows the Lotka-Volterra model of population dynamics implemented in AMIDOL using the predator-prey palette of nouns and verbs. This visual diagram is a recreation of semi-formal drawings a domain expert might draw, and meant to be more accessible than the ordinary differential equations which represent this system,

$$\frac{dx}{dt} = \alpha x - \beta xy \tag{9}$$

$$\frac{dy}{dt} = \delta xy - \gamma y \tag{10}$$

Where x is the number of rabbits, and y is the number of foxes. The AMIDOL translates this drawing automatically into the IR, and from the IR into a SciPy and odeint based representation,

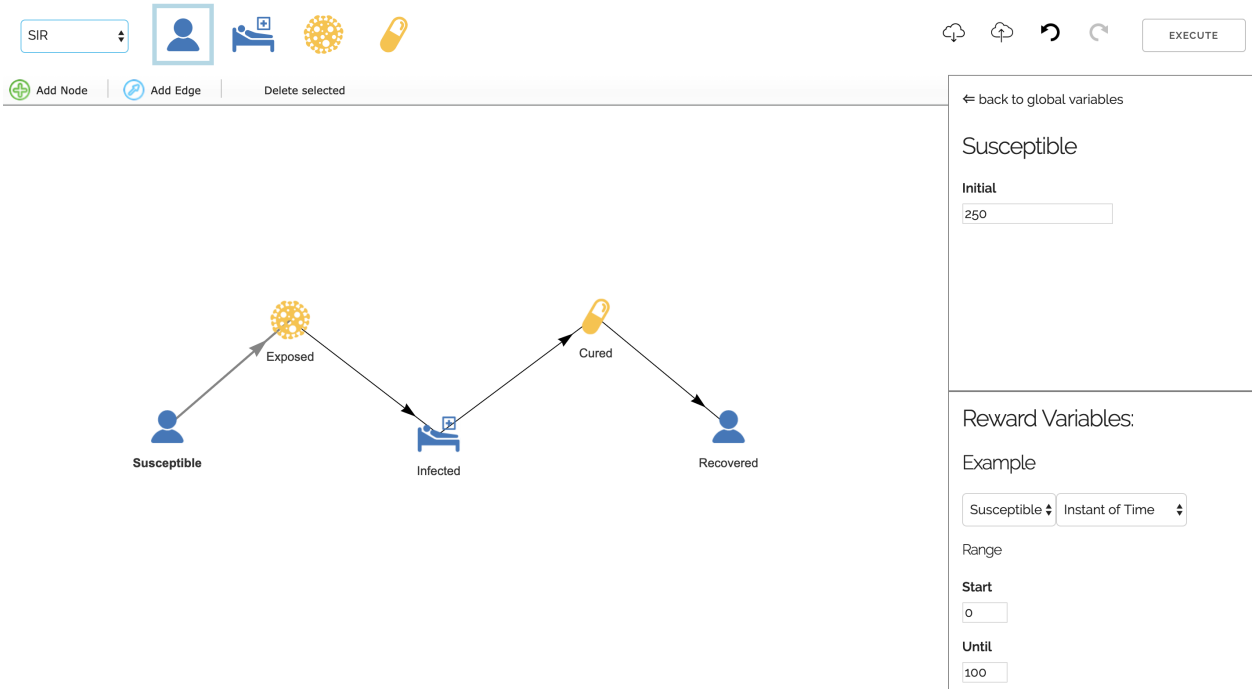


Figure 8: Current Front-End featuring the SIR VDSOL. The front-end enables the embedding of custom SVGs for nouns and verbs, allowing eventual specialization of produced figures, better matching current domain conventions.

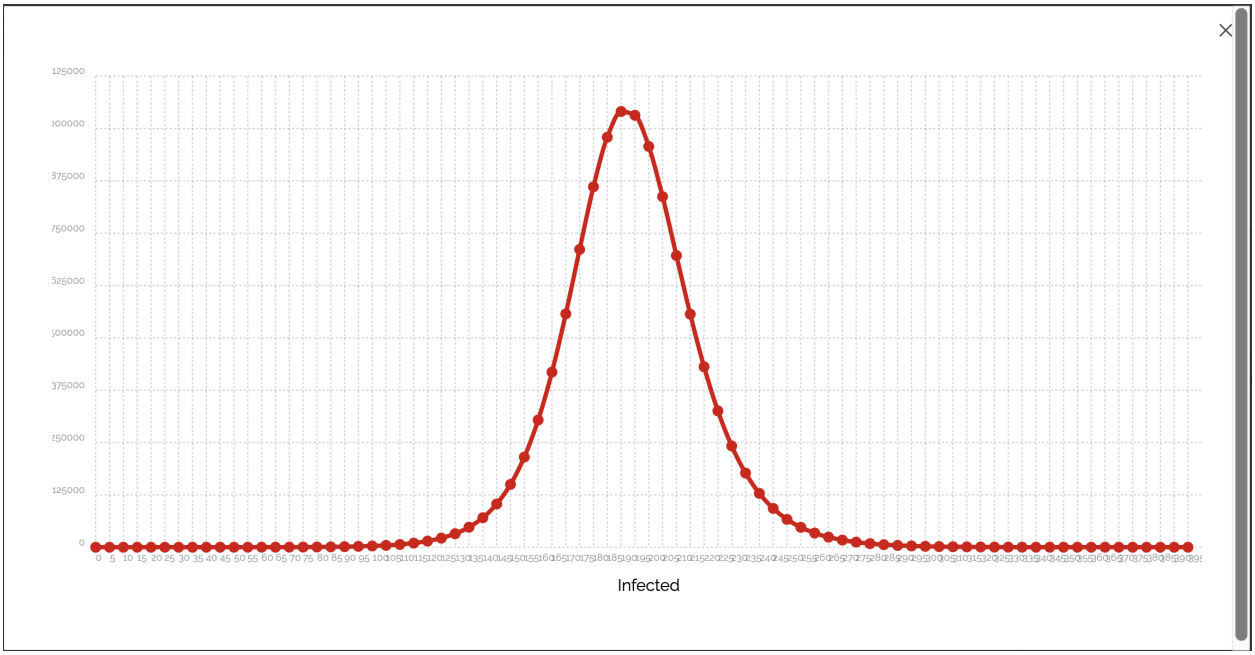


Figure 9: Output from the SIR Model in AMIDOL

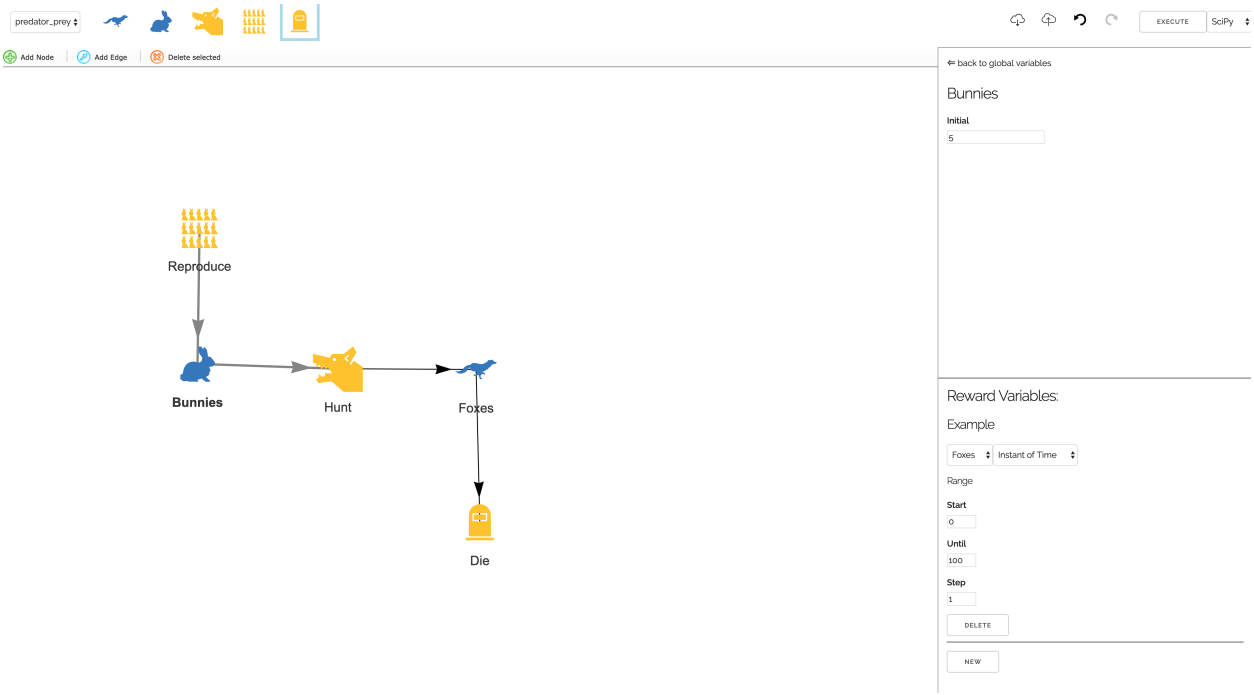


Figure 10: Example Lotka-Volterra Model in AMIDOL.

resulting in the following executable code:

```
  from scipy.integrate import odeint
import json
import numpy as np
import matplotlib.pyplot as plt

class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# User defined constants
Example = 0.0

# The ODE system
def deriv_(y_, t_):
    Bunnies, Foxes = y_
    dBunnies_ = 0.0 + 0.66 * Bunnies - 1.0 * Bunnies * Foxes
    dFoxes_ = 0.0 + 4.0 / 3.0 * Bunnies * Foxes - 1.0 * Foxes
    return dBunnies_, dFoxes_

# Boundary conditions and setup
timeRange_ = [ 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0,
11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0,
```

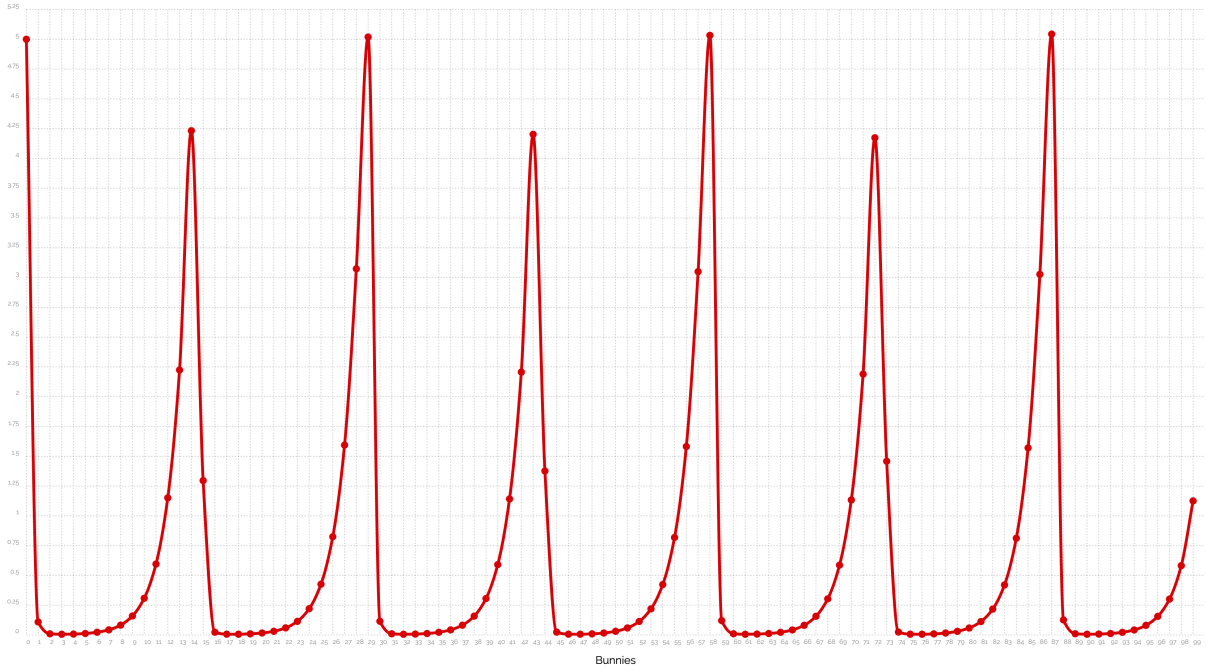


Figure 11: Output from the Lotka-Volterra in AMIDOL

```
22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0,
33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 42.0, 43.0,
44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0,
55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.0, 65.0,
66.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0,
77.0, 78.0, 79.0, 80.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0,
88.0, 89.0, 90.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0,
99.0 ]
y0_ = 5.0, 1.0
output = odeint(deriv_, y0_, timeRange_).T

print(json.dumps(output, cls=NumpyEncoder))
```

Figure 11 shows the current output of the AMIDOL inference engine, matching expectations for the Lotka-Volterra model as parameterized in this example.

A note on the results from the AMIDOL backend The Phase 1 prototype currently focuses on a proof of concept for the necessary core technologies of the AMIDOL framework. During Phase 2 we will be building on the backend's capabilities by implementing a results database and design of experiments interface. Part of this interface will include extensible plotting and visualization methods given a general form of results. An early part of our Phase 2 effort will be spent defining a formal results schema and representation, and selecting an appropriate results database to store this schema. Our design will focus, again, on generalizable techniques that apply to the abstract representations of models, but enable the user to interact with them only through knowledge of the

higher level VDSOL concepts.

9.2 Backend Architecture for IR and Inference Engine

The backend component interfaces with the UI via a local web server. The idea here is that every time the user interacts with the UI either to modify or to query the model, that information also gets relayed to the backend via a set of web endpoints. When the backend receives new information about the model from the UI, it parses this into some internal representation. This includes such tasks as parsing equations from user-inputted strings and checking that state variables being referred to actually exist. Queries submitted about a model follow a slightly longer path: after being parsed out and validated, the backend figures out how to transform the IR and the query into executable code using reward variables, executes this code, and returns the result back out to the user and eventual results database.

Within the backend, the IR is stored in a graph format resembling what the user constructed in the UI. By maintaining some degree of similarity, we hope to make it simpler to translate results obtained in the backend back out into something end users can easily understand. Queries about models are translated into code artifacts targeting existing solver and simulation programs. We wish to avoid doing actual simulations within the system, instead focusing on intelligently compiling queries into programs that external solvers can run allowing code reuse, and allowing the backend to generate targets for existing, high performance, solver engines. In order to do this we still need to implement a minimum amount of symbolic algebra (e.g. detecting when a continuous rate model is linear). For the time being, we've been targeting Python's SciPy module as a backend to answer basic simulation questions such as: the the initial value problem for general systems as well as for continuous-time Markov chains.

The backend is written in Scala, leveraging a set of libraries built on top of the Akka actor system for the web server, JSON processing, asynchronous computation, and eventually for the graph in which the IR and data will be stored. The advantages to using Scala include: deployable anywhere the JVM runs, large set of available libraries, and a functional outlook which lends itself well to compiler problems.

9.3 Integrating AMIDOL

The JSON API which defines client/server interaction is the only channel by which tangible VDSOL models and the computational AFI/IR communicate. The UI itself doesn't hold any persistent model state which is not represented in a given VDSOL; it is precisely a view or translation of the AFI/IR. This property is what enables the list of integration concepts, and restful architecture.

10 Next Steps and Phase 2

The Phase 1 prototype for AMIDOL proves the core technologies and functionality required for success in this project are feasible, achievable, and provide the necessary functionality to support our vision for Phase 2. Phase 2 will work on extending this base functionality and building on top of our current architecture. During Phase 2 we will work with more complex models, including full models of H3N2 and H5N1 which include multiple geographic compartments, and the calculation of complex reward variables on these models which require a fully developed Design of Experiments interface as well as model composition.

A large component of our Phase 2 progress will revolve around the use of the results database as

a support for tying the results of solving reward variables back to the primitives present in the VDSOL. A key lesson learned from Phase 1 was the importance of relating reward variables to elements in the VDSOL. The current state of the art for reward variables involves their definition on model primitives such as state variables and events, which are elements of our IR and not a given VDSOL. This is not ideal for our goals and desired outcomes which focus on expert interaction with the VDSOL while hiding details of the underlying model. Our intention is to extend the definition of reward variables in Phase 2 such that we have equivalents for rate and impulse rewards which are instead defined more naturally over nouns and verbs. We plan to do so by defining reward coupling points in an extension of our VDSOL which indicate what a measure on a noun or a verb will return. This will allow rewards to be more naturally defined than they currently are both in our prototype and by other modeling tools, and will allow us to infer information on **structural types** of nouns and verbs, determine equivalences, and capture knowledge from domain experts when they tell us they believe nouns in two different models represent the same abstract quantity.

We intend to collaborate with other performers in the ASKE program, especially TA1 performers, to identify natural ways to import diagrams and models extracted from primary sources, and translate them into AMIDOL models. This will help show the expressive power of AMIDOL's system of model definition, and the suitability of the AMIDOL backend and IR as a universal model translation, optimization, and representation layer.

References

[1] Karsten Ahnert and Mario Mulansky. Odeint–solving ordinary differential equations in c++. In *AIP Conference Proceedings*, volume 1389, pages 1586–1589. AIP, 2011.

[2] Robert Bailis, Majid Ezzati, and Daniel M Kammen. Mortality and greenhouse gas impacts of biomass and petroleum energy futures in africa. *Science*, 308(5718):98–103, 2005.

[3] Fred Brauer, Carlos Castillo-Chavez, and Carlos Castillo-Chavez. *Mathematical models in population biology and epidemiology*, volume 40. Springer, 2012.

[4] Kim B Bruce. *Foundations of object-oriented languages: types and semantics*. MIT press, 2002.

[5] CDC. National, regional, and state level outpatient illness and viral surveillance. https://www.cdc.gov/flu/weekly/fluactivitysurv.htm. Accessed: January 2019.

[6] Giovanni Chiola, Marco Ajmone Marsan, Gianfranco Balbo, and Gianni Conte. Generalized stochastic petri nets: A definition at the net level and its implications. *IEEE Transactions on software engineering*, 19(2):89–107, 1993.

[7] Gianfranco Ciardo and Robert Zijal. Well-defined stochastic petri nets. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1996. MASCOTS'96., Proceedings of the Fourth International Workshop on*, pages 278–284. IEEE, 1996.

[8] Daniel D Deavours and William H Sanders. An efficient well-specified check. In *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, pages 124–133. IEEE, 1999.

[9] David Fisman, Edwin Khoo, and Ashleigh Tuite. Early epidemic dynamics of the west african 2014 ebola outbreak: estimates derived with a simple two-parameter model. *PLoS currents*, 6, 2014.

[10] Christophe Fraser, Christl A Donnelly, Simon Cauchemez, William P Hanage, Maria D Van Kerkhove, T Déirdre Hollingsworth, Jamie Griffin, Rebecca F Baggaley, Helen E Jenkins, Emily J Lyons, et al. Pandemic potential of a strain of influenza a (h1n1): early findings. *science*, 2009.

[11] Herbert I Freedman. *Deterministic mathematical models in population ecology*, volume 57. Marcel Dekker Incorporated, 1980.

[12] Roberto Freire. A technique for simulating composed san-based reward models. 1990.

[13] Eric L Haseltine and James B Rawlings. Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *The Journal of chemical physics*, 117(15):6959–6969, 2002.

[14] Frank Hoppensteadt. Predator-prey model. *Scholarpedia*, 1(10):1563, 2006.

[15] Ronald A Howard. *Dynamic probabilistic systems: Markov models*, volume 1. Courier Corporation, 2012.

[16] Kai Lampka and Markus Siegle. Symbolic composition within the möbius framework. In *Proc. of the 2nd MMB Workshop*, pages 63–74, 2002.

[17] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[18] Max D Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

[19] Ali Movaghar. Performability modeling with stochastic activity networks. 1985.

[20] James M Murphy, David MH Sexton, David N Barnett, Gareth S Jones, Mark J Webb, Matthew Collins, and David A Stainforth. Quantification of modelling uncertainties in a large ensemble of climate change simulations. *Nature*, 430(7001):768, 2004.

[21] Travis E Oliphant. Scipy tutorial, 2004.

[22] BG Olivier. Pysces cbmpy: Constraint based modelling in python, 2011.

[23] Brett G Olivier, Johann M Rohwer, and Jan-Hendrik S Hofmeyr. Modelling cellular systems with pysces. *Bioinformatics*, 21(4):560–561, 2005.

[24] Muhammad A Qureshi, William H Sanders, Aad PA Van Moorsel, and Reinhard German. Algorithms for the generation of state-level representations of stochastic activity networks with general reward structures. *IEEE Transactions on Software Engineering*, 22(9):603–614, 1996.

[25] William H Sanders and Luai M Malhis. Dependability evaluation using composed san-based reward models. *Journal of parallel and distributed computing*, 15(3):238–254, 1992.

[26] William H Sanders and John F Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[27] William H Sanders and John F Meyer. Stochastic activity networks: Formal definitions and concepts. In *School organized by the European Educational Forum*, pages 315–343. Springer, 2000.

[28] William H. Sanders, W Douglas Obal II, Muhammad A. Qureshi, and FK Widjanarko. The ultrasan modeling environment. *Performance Evaluation*, 24(1-2):89–115, 1995.

[29] William Harry Sanders. Construction and solution of performability models based on stochastic activity networks. 1988.

[30] Ranjan Srivastava, L You, J Summers, and J Yin. Stochastic vs. deterministic modeling of intracellular viral kinetics. *Journal of theoretical biology*, 218(3):309–321, 2002.

[31] Leor S Weinberger, John C Burnett, Jared E Toettcher, Adam P Arkin, and David V Schaffer. Stochastic gene expression in a lentiviral positive-feedback loop: Hiv-1 tat fluctuations drive phenotypic diversity. *Cell*, 122(2):169–182, 2005.

A Additional Domain Models

During the course of our work for Phase 1, and in preparation for Phase 2, we have begun identifying and specifying a number of additional models which will help us in our development goals, and in validating our implementations. Those models are documented in this appendix, but not implemented in our Phase 1 prototype.

A.1 Artificial Chemistry and Intracellular Viral Infection

We have also introduced two simple models of artificial chemistry, and viral infection first introduced by [30, 13] to test different VDSOLs with similar solution techniques. While simple, models such as the crystallization model:

$$2A \xrightarrow{e_1} B \tag{11}$$

$$A + C \xrightarrow{e_2} D \tag{12}$$

Allow us to test features of the UI for AMIDOL in a different domain, and the ability of our predicates to handle non-conservation of value in state variables, along with enabling conditions due to the presence of non-renewable reactions.

Figure 12 shows a more complex version of the crystallization model with catalysts, which models viral replication using a chemical kinetic model.

$$\text{nucleotides} \xrightarrow{e_1} \text{genome} \tag{13}$$

$$\text{nucleotides} + \text{genome} \xrightarrow{e_2} \text{template} \tag{14}$$

$$\text{nucleotides} + \text{aminoacids} \xrightarrow{e_3} \text{struct} \tag{15}$$

$$\text{template} \xrightarrow{e_4} \text{degraded} \tag{16}$$

$$\text{struct} \xrightarrow{e_5} \text{secreted} \tag{17}$$

$$\text{struct} \xrightarrow{e_5} \text{degraded} \tag{18}$$

$$\text{genome} + \text{struct} \xrightarrow{e_6} \text{virus} \tag{19}$$

$$\tag{20}$$

This model also features competing events, allowing us to further test our backend's capability for dealing with cases of multiple enabled events.

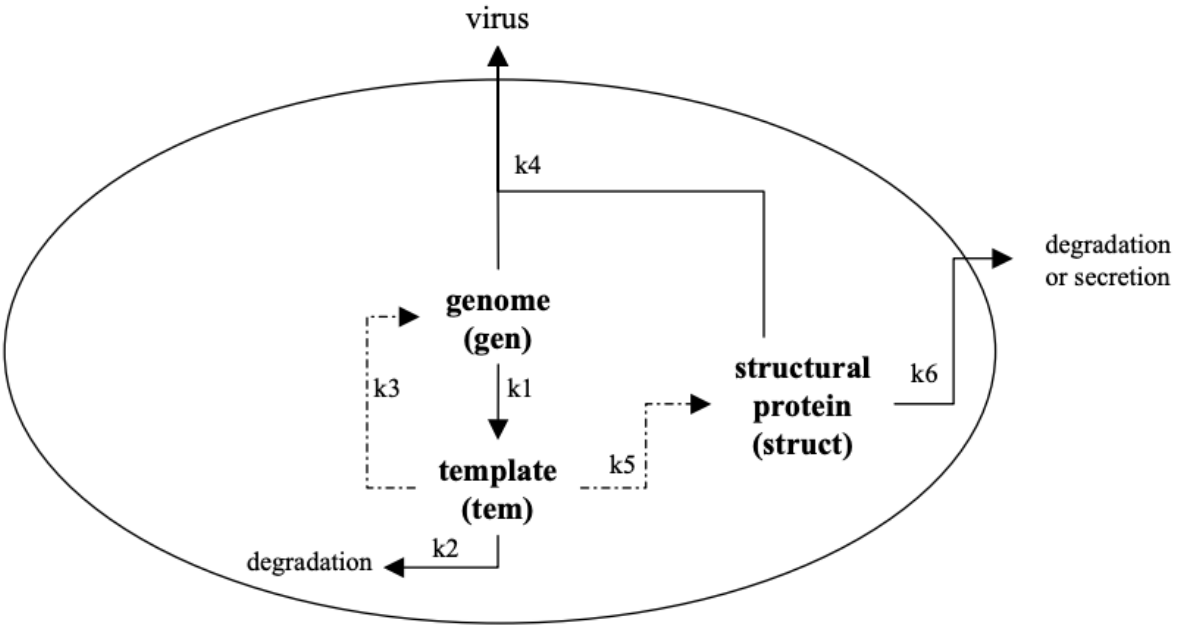


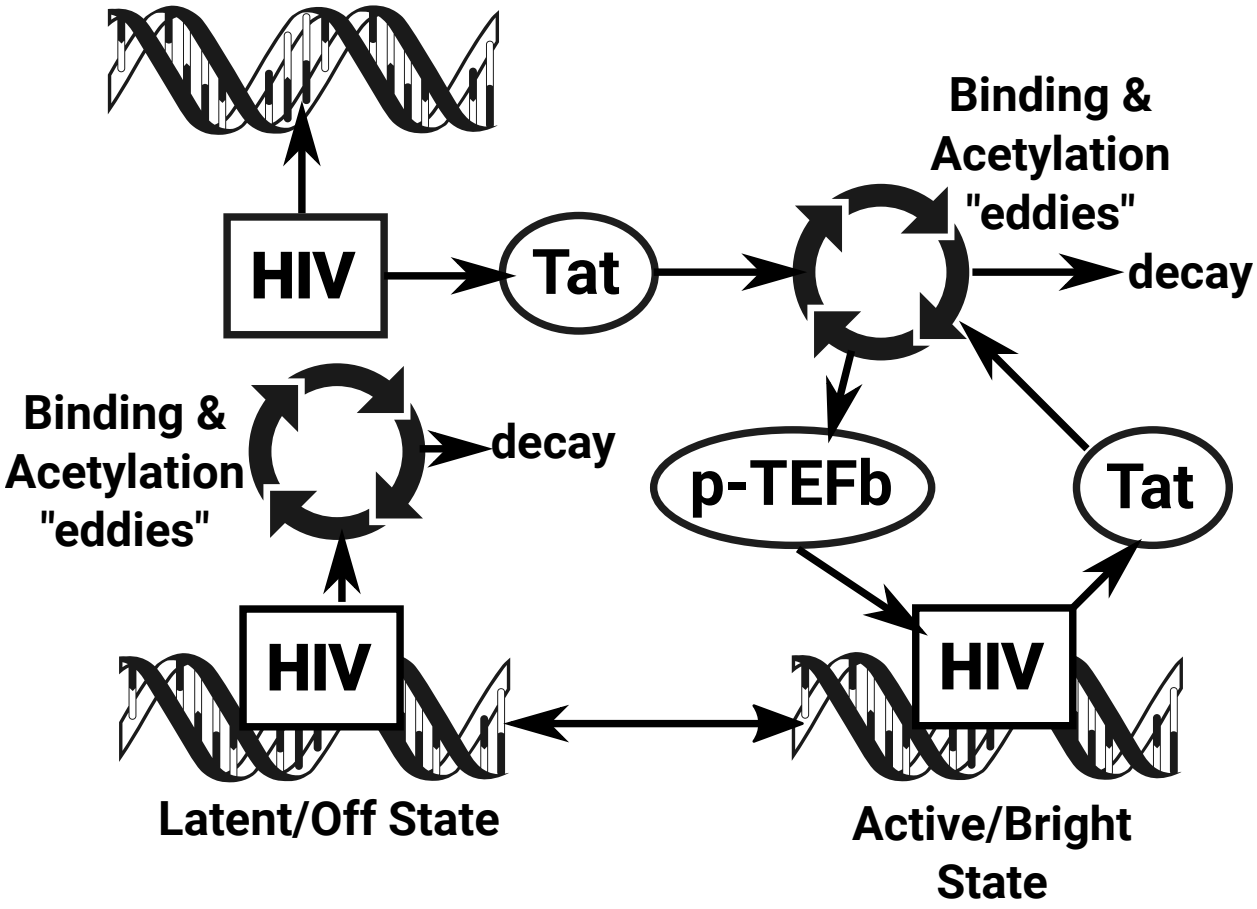
Figure 12: Model of viral replication cycle with catalytic reactions

A.2 HIV Transactivation Model

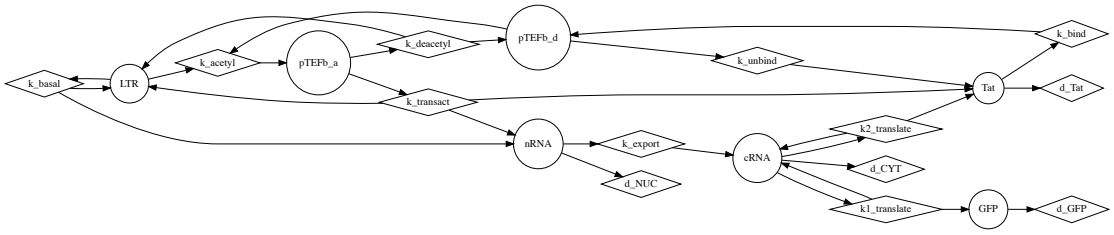
We also employ the HIV Transactivation model presented in [31] as a more complex model of an epidemic process, with the added challenge of handling a system which is stiff, and thus difficult to solve efficiently. The base model is shown in Figures 13a and 13b

Note the use of multiple "Tat" symbols in Figure 13a. Sometimes scientists draw the same symbol multiple places as an "alias" for the same underlying state variable, which has added the requirement to AMIDOL of supporting entity resolution. Our goal with AMIDOL is not to fundamentally restrict a domain scientist when drawing diagrams, but rather to assist them in their natural process, and allow them to use the same conventions they are already familiar with, and which exist in their field.

The equations governing this model are given by:



(a) Semi-formal diagram of the molecular model of the Tat transactivation circuit.



(b) Simple noun (circle) and verb (square) representation of Tat model without ambiguity and aliasing.

$$LTR \overset{k_{basal}}{\rightarrow} LTR + nRNA \tag{21}$$

$$nRNA \overset{k_{export}}{\rightarrow} cRNA \tag{22}$$

$$cRNA \overset{k1_{translate}}{\rightarrow} GFP + cRNA \tag{23}$$

$$cRNA \overset{k2_{translate}}{\rightarrow} Tat + cRNA \tag{24}$$

$$Tat \overset{k_{bind}/k_{unbind}}{\leftrightarrow} pTEFb_d \tag{25}$$

$$LTR + pTEFb_d \overset{k_{acetyl}/k_{deacetly}}{\leftrightarrow} pTEFb_a \tag{26}$$

$$pTEFb_a \overset{k_{transact}}{\leftrightarrow} LTR + nRNA + Tat \tag{27}$$

$$GFP \overset{d_{GFP}}{\rightarrow} \emptyset \tag{28}$$

$$Tat \overset{d_{Tat}}{\rightarrow} \emptyset \tag{29}$$

$$cRNA \overset{d_{CYT}}{\rightarrow} \emptyset \tag{30}$$

$$nRNA \overset{d_{NUC}}{\rightarrow} \emptyset \tag{31}$$

Using real data generated with flow cytometry, this model can be parameterized as follows:

$$k_basal = 10^{-8}(transcripts/s) \tag{32}$$

$$k_export = 0.00072(1/s) \tag{33}$$

$$k1_translate = 0.5(1/s) \tag{34}$$

$$k2_translate = 0.005(1/s) \tag{35}$$

$$k_bind = 10^{-4}(1/(mol * s)) \tag{36}$$

$$k_unbind = 10^{-2}(1/s) \tag{37}$$

$$k_acetyl = 10^{-3}(1/(mol * s)) \tag{38}$$

$$k_deacetyl = 0.9(1/s) \tag{39}$$

$$k_transact = 0.1(1/s) \tag{40}$$

As can be seen, the rates differ by many orders of magnitude, a situation that causes the model to be classified as stiff, and thus difficult to solve by many ODE techniques. We are using this model to test the Design of Experiments capability of AMIDOL to automatically select appropriate solution techniques for a user, avoiding numerical instability and the use of inefficient algorithms which otherwise could be pitfalls for traditional modeling methods.