ASKE Milestone 2 for AMIDOL

Eric Davis[1], Alec Theriault[1], Max Orhai[1], Eddy Westbrook[1], and Ryan Wright[1]

[1]Galois, Inc

1 Introduction

Complex system analysis currently requires teams of domain experts, data scientists, mathematicians, and software engineers to support the entire life cycle of model-based inference. The models that result are often bespoke, lack generalizability, are not performable, and make it difficult to synthesize actionable knowledge and policies from their raw outputs. In this report we describe the current prototype system for AMIDOL: the Agile Metamodel Interface using Domain-specific Ontological Languages, a project that aims to reduce the overhead associated with the model life cycle and enables 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 discuss the current design principles of the AMIDOL prototype, its capabilities, plans for development, and formal aspects of the system.

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's abstract representations are executable, allowing AMIDOL's inference engine to execute prognostic queries on reward models and communicate results to domain experts. AMIDOL binds 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.

Our techniques for achieving these goals incorporate abstract functional representations, intermediate languages, and semantic knowledge representation and binding in graph structures into traditional machine learning and model solution techniques.

2 VDSOL Definition

AMIDOL is designed to support the definition of ontological languages which describe systems as formal objects. Objects for a given domain are organized into *toolkits* 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.

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.

VDSOLs in AMIDOL are constructed using a graphical user interface implemented using asynchronous javascript and XML to build a responsive interface to define models of complex systems, reward models used to explore and understand complex system behavior, and to interact with the results of solvers implemented in the Machine-Assisted Inference Engine.

2.1 Basic Language Properties

Nouns : Nouns in AMIDOL represent portions of the model associated with its state space. The AMIDOL IR translates noun elements into state variables and constants. 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 associated with the noun.

Verbs : Verbs in AMIDOL represent activities or events which can occur in a model, and which act upon nouns changing the state of the system. Verbs are associated with a few mandatory and optional properties which impact their translation into the intermediate representation. All verbs have a mandatory rate which defines the rate at which the associated event occurs. This rate can be dependent on nouns in the model. Verbs have an optional enabling condition which can be dependent on nouns in the model. The enabling condition defines that state variable bounds during which the associated event is enabled, and can be specified as an algebraic expression over state variables associated with nouns in a model. Verbs also have an optional output function which defines which nouns are impacted when the associated event fires.

2.2 Composability of Atomic Models

AMIDOL is being designed to support the composition of individual models 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* [13, 17] and *event-synchronization* [10].

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 is planned to support both replicate and join operations defined over Petri-nets and stochastic activity networks. [16] 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 [1], which allows users to specify abstract models with undefined states and events.

3 Abstract 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.

The intermediate representation employed by AMIDOL has its roots in Markov models [9], Generalized Stochastic Petri-nets with inhibitor arcs [3] which have been shown to be Turing complete, and stochastic activity networks [11, 15] which are extensions 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.

3.1 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 $SandE$ 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 inhibtor 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.

4 Inference Engine

ODE Solver :

Numerical Solution :

Discrete Event Simulation :

5 Reward Variables and Reward Models

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).

[12, 5, 4, 14]

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.

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 [8] 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:

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

where

- $J_{[t,t+1]}^{\nu}$ 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 $[t, t+1]$.
- $I_{t\to\infty}^{e}$ 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.

5.4 Expressions on Reward Variables

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

AMIDOL aims to support a number of complex queries and prognostics, as shown in Figure 1. To support a rich interface, we will make use of a Design of Experiments interface coupled with a results database.

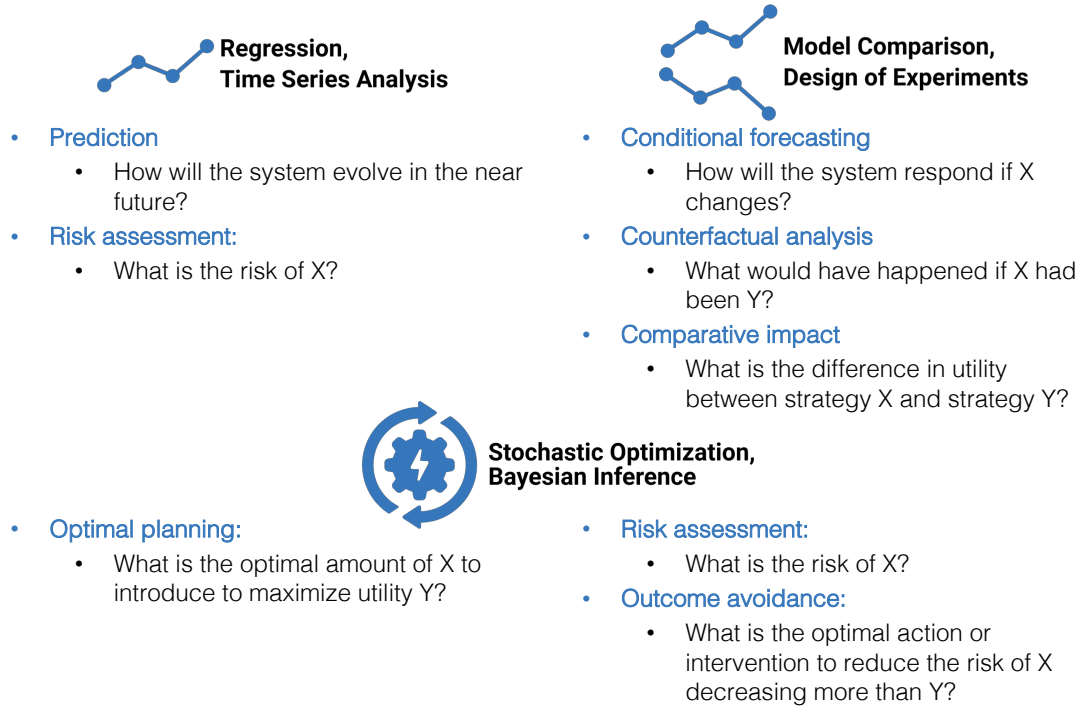


Figure 1

6.1 Prognostic Queries

6.2 Model Comparison

6.3 Design of Experiments

6.4 Conterfactural Exploration, Planning, Crisis Response

6.5 Correctness and Uncertainty

6.6 Communication of Results

7 Domain Models

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

7.1 SIS/SIRS

H1N1 R_0 importance [7].

Ebola R_0 importance [6]

CDC Data [2]

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 utlizes 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

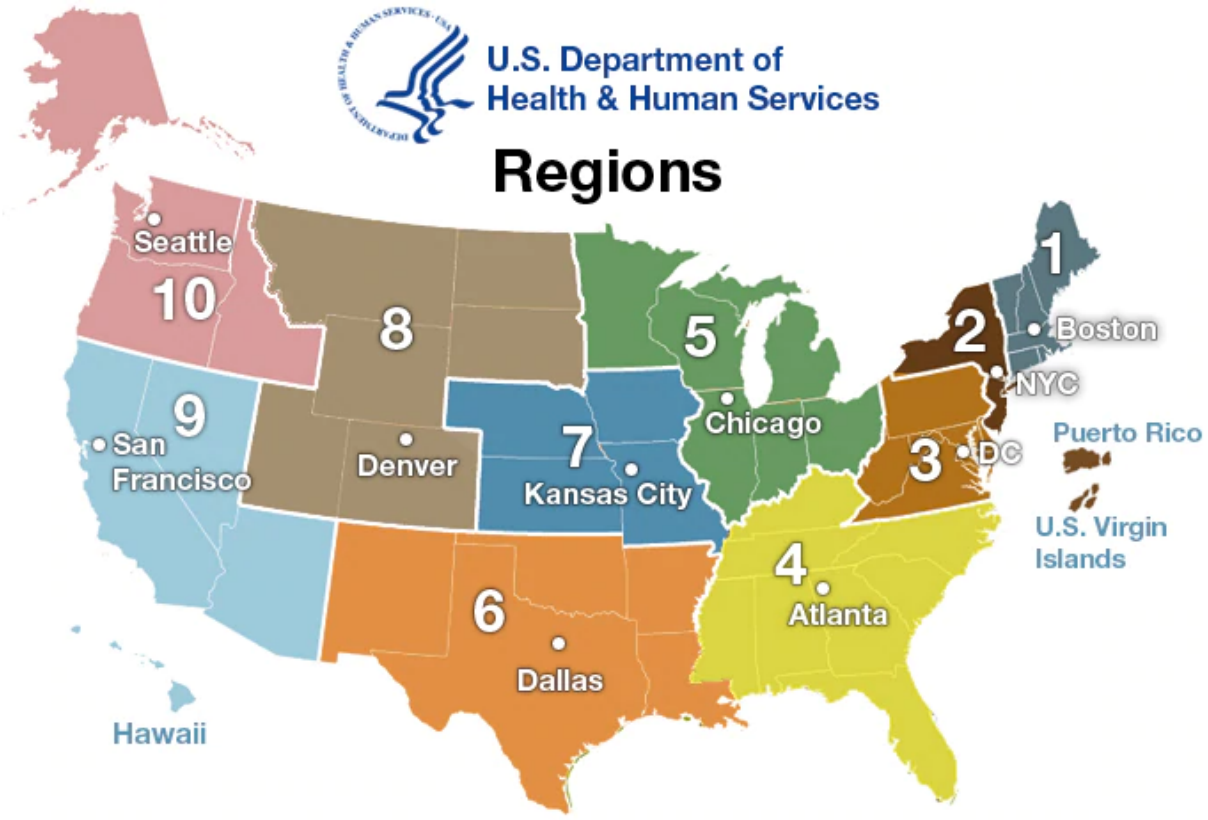


Figure 2: Department of Human and Health Services designated regions.

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.

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 will be working 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 will estimate R0 for the epidemic curve as shown in Figure 3

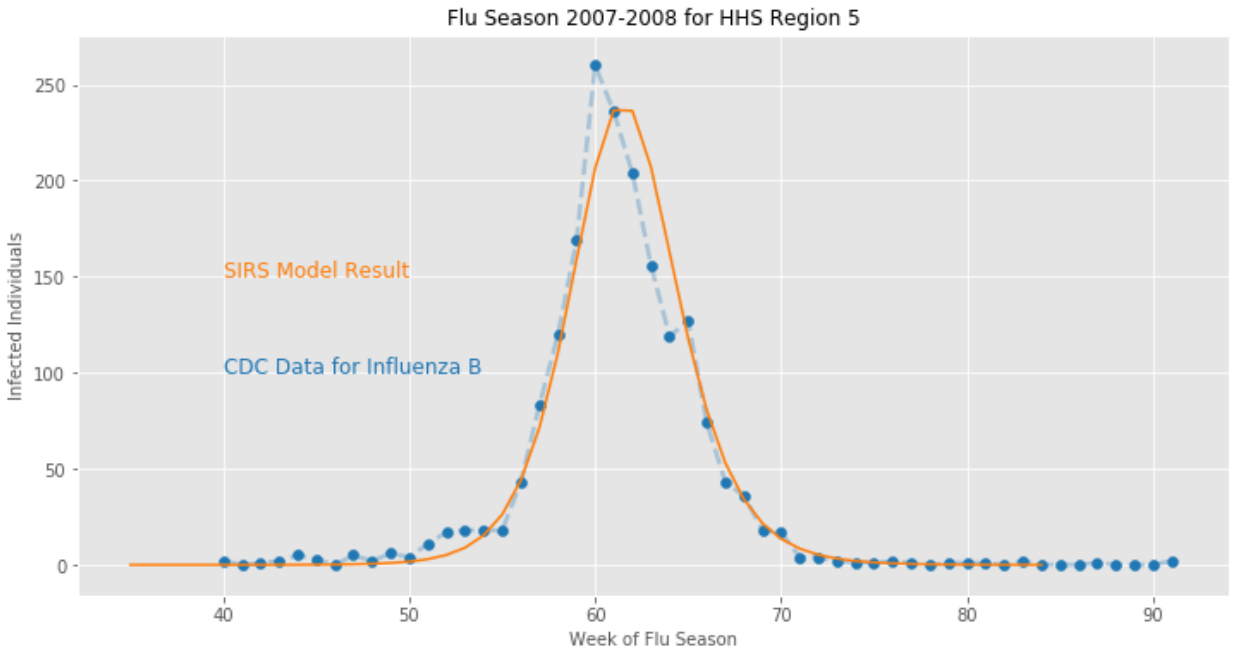


Figure 3: 2007 - 2008 Flu Season

7.2 Artificial Chemistry

7.3 Viral Infection Model

Note the use of multiple "Tat" symbols in Figure 4a. Sometimes scientists draw the same symbol multiple places as an "alias" for the same underlying state variable.

$$LTR \overset{k_{basal}}{\rightarrow} LTR + nRNA \tag{1}$$

$$nRNA \overset{k_{export}}{\rightarrow} cRNA \tag{2}$$

$$cRNA \overset{k1_{translate}}{\rightarrow} GFP + cRNA \tag{3}$$

$$cRNA \overset{k2_{translate}}{\rightarrow} Tat + cRNA \tag{4}$$

$$Tat \overset{k_{bind}/k_{unbind}}{\leftrightarrow} pTEFb_d \tag{5}$$

$$LTR + pTEFb_d \overset{k_{acetyl}/k_{deacetly}}{\leftrightarrow} pTEFb_a \tag{6}$$

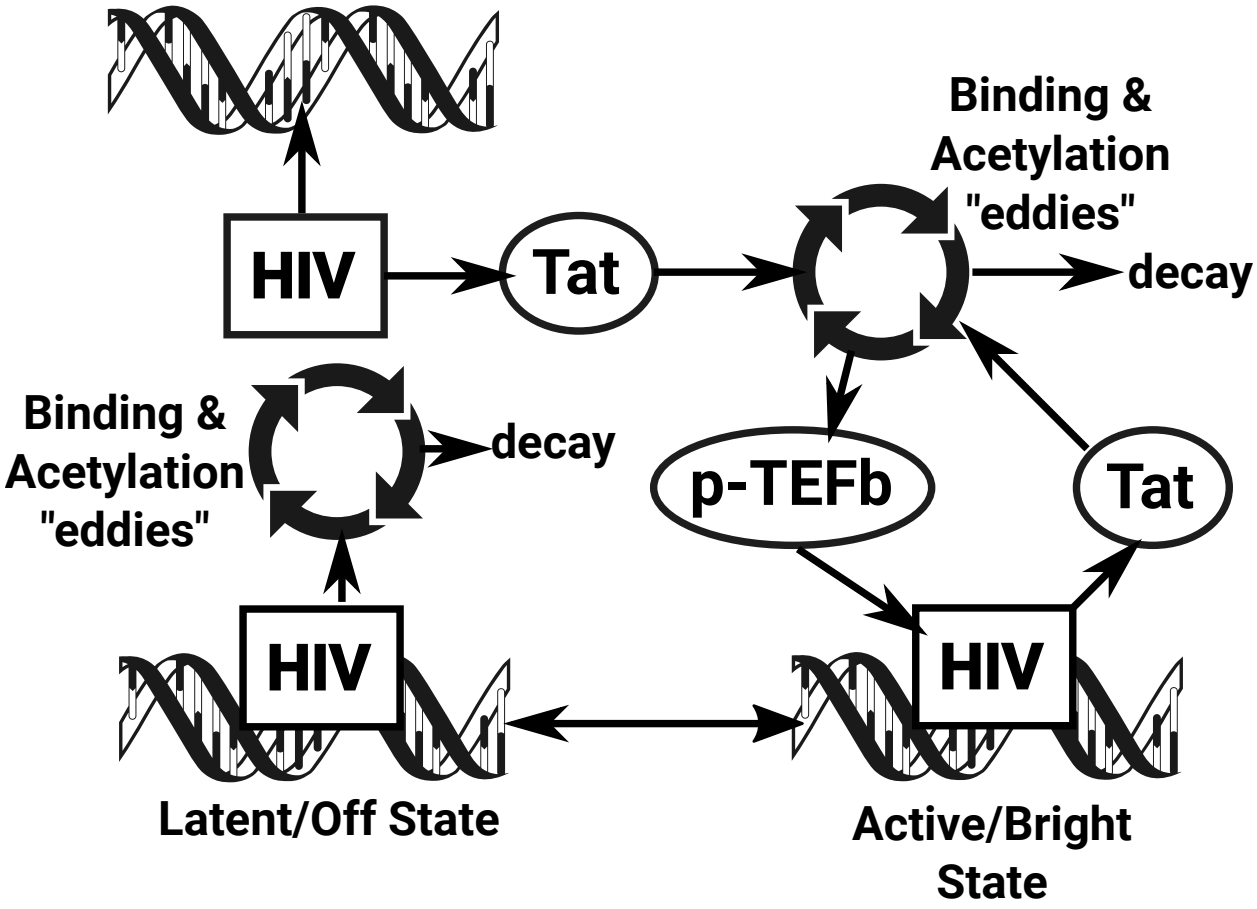
$$pTEFb_a \overset{k_{transact}}{\leftrightarrow} LTR + nRNA + Tat \tag{7}$$

$$GFP \overset{d_{GFP}}{\rightarrow} \emptyset \tag{8}$$

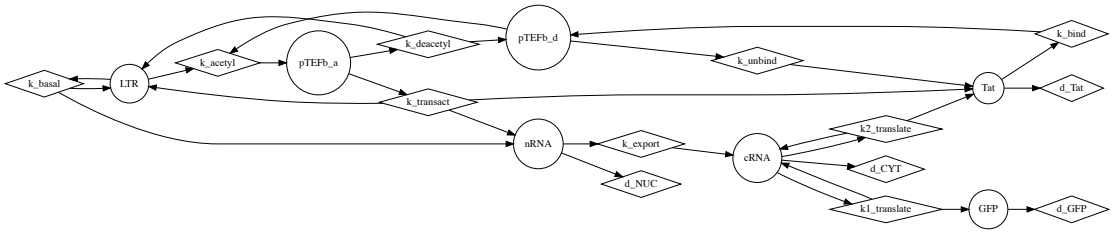
$$Tat \overset{d_{Tat}}{\rightarrow} \emptyset \tag{9}$$

$$cRNA \overset{d_{CYT}}{\rightarrow} \emptyset \tag{10}$$

$$nRNA \overset{d_{NUC}}{\rightarrow} \emptyset \tag{11}$$



(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.

7.4 H5N1 Model

7.5 H3N2 Model

8 User Stories

9 Code Repositories and Current Builds

The UI (an Elm project contained in the ‘ui‘ directory) is a series of HTML pages through which users will interact with the system. That includes drawing/manipulating semi-formal models, issuing queries, loading data.

The main system (a Scala project contained in the ‘ir‘ directory) compiles questions that users may have about models they’ve created into code artifacts targeting different simulation/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).

9.1 Front-End Architecture for VDSOLs

Amidol’s user interface is a small collection of web pages which communicate with the Scala back-end using JSON over HTTP. This client/server approach leverages standard browser technologies like HTML5, CSS, and SVG graphics 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.

The Javascript code which implements the user interface is compiled from Elm, a strongly typed functional language designed specifically for front-end web development. Elm’s runtime system is similar to that of popular Javascript frameworks like React, but the language provides some distinct advantages in terms of correctness, performance, and ease of refactoring. In addition to the capabilities provided by the Elm package ecosystem, Elm applications can interoperate directly with Javascript, allowing Amidol to make use of the best available web-based data visualization libraries.

Integration concepts: - API schema and versioning - supports collaboration between multiple users - one server, potentially multiple client implementations - persistent server, ephemeral clients - resource-heavy servers (cluster?), lightweight clients

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. We are in the process of designing a set of user interactions with these graph models, including save / load, model composition, and various forms of simulation analysis and questioning. The UI will also show the answers to users’ queries in appropriate visual forms, such as multivariate time-series plots.

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 ask questions of 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. Questions asked of the model follow a slightly longer path: after being parsed out and validated, the backend figures out how to transform the IR and the question into executable code, executes this code, and returns the result back out to the user.

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. Questions asked 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. In order to do this sort of thing, we still do need to implement a minimum amount of symbolic algebra (ex: 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

I think it's worth saying somewhere that the JSON API which defines client/server interaction is the only channel by which tangible VDSOL models and computational AFI communicate. In other words, the UI doesn't hold any persistent model state which is not represented in the API; it is precisely a view or translation of the AFI/IR. This property is what enables the list of integration concepts. In a word, it's a RESTful architecture.

10 Roadmap for Future Development

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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,

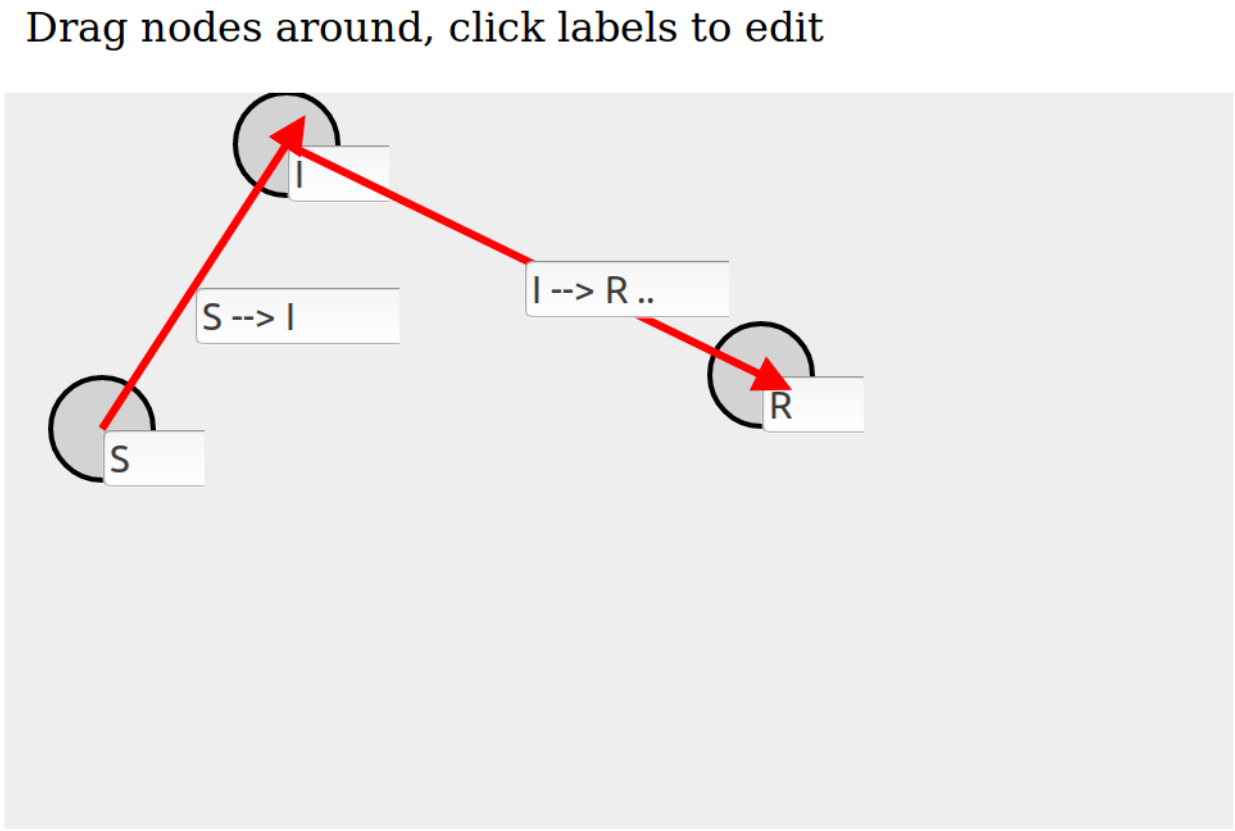


Figure 5

```
bash-3.2$ pwd
/Users/atheriault/Code/AMIDOL/ir
bash-3.2$ curl -H "Content-Type: application/json" \
>              -X POST \
>              -d @src/main/resources/sirs_graph.json \
>              "http://localhost:8080/appstate/model" -w "\n"
Model has been updated
bash-3.2$ curl -H "Accept: application/json" \
>              -X GET \
>              "http://localhost:8080/appstate/model" -s | jq '.'
{
  "links": [
    {
      "id": -1,
      "label": "β * Susceptible * Infectious / N",
      "source": 0,
      "target": 1
    },
    {
      "id": -2,
      "label": "γ * Infectious",
      "source": 1,
      "target": 2
    },
    {
      "id": -3,
      "label": "μ * Recovered",
      "source": 2,
      "target": 0
    }
  ],
  "nodes": [
    {
      "id": 0,
      "label": "Susceptible",
      "location": {
        "x": 100,
        "y": 100
      },
      "view": "susceptible_dummy.svg"
    },
    {
      "id": 1,
      "label": "Infectious",
      "location": {
        "x": 300,
        "y": 100
      },
      "view": "infectious_dummy.svg"
    },
    {
      "id": 2,
      "label": "Recovered",
      "location": {
        "x": 500,
        "y": 100
      },
      "view": "recovered_dummy.svg"
    }
  ]
}
bash-3.2$ exit
```

Figure 6

```
bash-3.2$ pwd
/Users/atheriault/Code/AMIDOL/ir
bash-3.2$ cat query.json | jq
{
  "constants": {
    "γ": 0.333,
    "β": 0.413,
    "μ": 0
  },
  "boundary": {
    "Susceptible": 51999999,
    "Infectious": 1,
    "Recovered": 0
  },
  "initialTime": 40,
  "finalTime": 400,
  "stepSize": 1,
  "savePlot": "img.png"
}
bash-3.2$ curl -H 'Accept: application/json' \
>              -X GET -G \
>              --data-urlencode "inputs=$(<query.json)" \
>              "http://localhost:8080/backends/scipy/integrate"
bash-3.2$ exit
```

Figure 7

Emily J Lyons, et al. Pandemic potential of a strain of influenza a (h1n1): early findings. *science*, 2009.

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

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

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

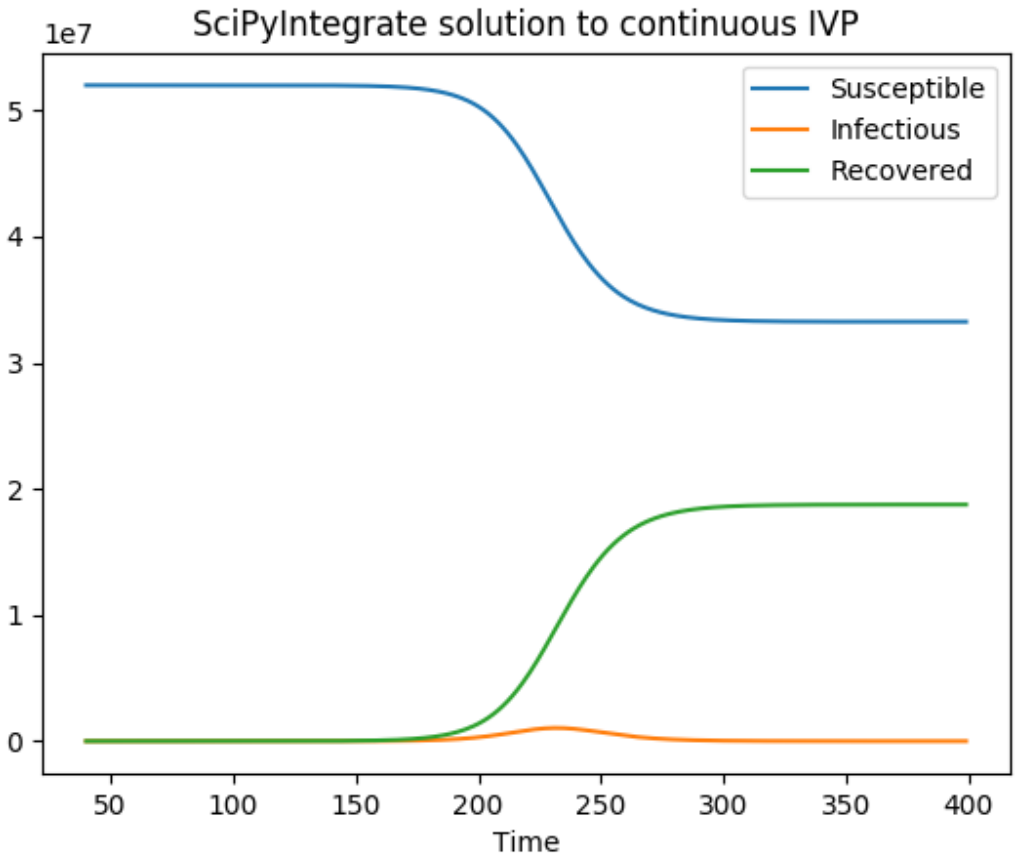

SciPyIntegrate solution to continuous IVP
1e7
5
4
3
2
1
0
50
100
150
200
250
300
350
400
Time
Susceptible
Infectious
Recovered

Figure 8

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