# Machine-Assisted Extraction of Formal Semantics from Domain Specific Semi-Formal Diagrams

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

[1]Galois, Inc

**Abstract**

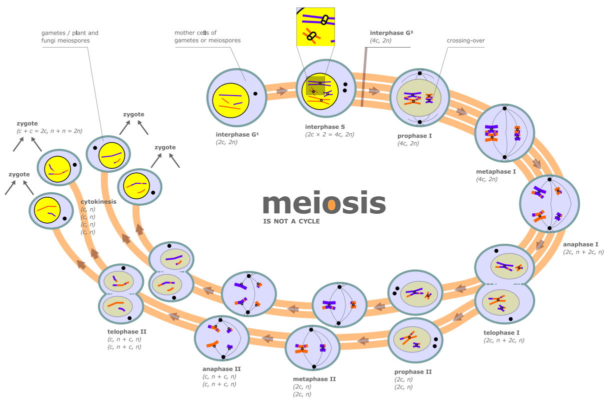
Analyzing complex systems is a challenging process which requires not only teams of domain experts but often also a multidisciplinary team of data scientists, mathematicians, statisticians, and software engineers in order to support the life cycle of model development, model-based inference, as well as knowledge extraction and synthesis. The models that typically result from this process today are bespoke, lack generalizability, are not performable, lack reusability, and make the task of synthesizing actionable knowledge and policies from their raw outputs difficult. In this paper we describe AMIDOL: the Agile Metamodel Inference using Domain-specific Ontological Languages, a project that aims to reduce the overhead associated with the development, deployment, maintenance, and reuse of models of complex systems. Our technique utilizes a common intermediate representation which is designed to support a number of scientific, physical, social, and hybrid domains by allowing domain experts to define their models in a novel way: using domain specific ontological languages (VDSOLs). The intermediate abstract representation provides formal, executable, meaning to the semi-formal diagrams domain experts normally create, and allows the inference engine to build prognostic queries on associated reward models. AMIDOL binds results from the inference engine to the original ontologies providing more explainability when compared to conventional methods.

## 1 Introduction

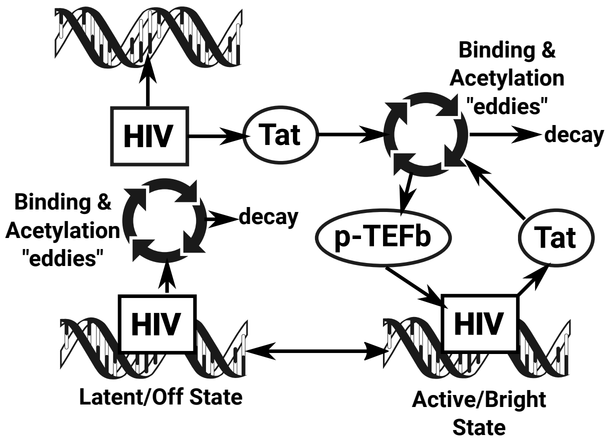
The construction of computational models is an important practice for scientists, engineers, policy makers, and other domain experts as it provides a way to make formal predictions based on a formal expectation of system behavior, and allows practitioners to test hypotheses and explain phenomena in complex systems. The process of building suitable models, however, is a laborious one which requires diverse teams of experts, significant manual effort, and typical results in models with severe limitations, low performability, and little reusability or generalizability. Such models are not only costly to produce, both in time and effort, but are also prone to errors, difficult to verify, and do not utilize best practices in software development.

AMIDOL seeks to improve the problems inherent to modern 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 techniques applied to achieve these goals incorporate abstract intermediate representations, semantic knowledge representation and binding in graph structures, traditional machine learning and model solution techniques, with intuitive model generation giving meaning to the semi-formal diagrams already being generated by domain experts.

Current modeling formalisms are often specified in formal languages which seem arcane and unnatural to domain experts, but have unambiguous formal meaning with defined execution semantics.



(a) Example of a semi-formal diagram of Meoisis. CC-BY-SA 3.0 Marek Kultys, July 2, 2008.



(b) Example of a semi-formal diagram of the molecular model of the Tat transactivation circuit.

Figure 1: Examples of semi-formal diagrams drawn by domain experts to represent operational semantics and complex system models.

By contrast domain experts have naturally developed visual semi-formal ways of describing the systems they study, such as those illustrated in Figure 1, but which lack semantic meaning, executability, and are often ambiguous. In Luca Cardelli's paper on the abstract machines of systems biology [2] he notes that these semi-formal diagrams come close to capturing operational semantics of process algebras, a concept he expanded on when deriving a formal calculus of membrane interactions [1]. Similarly, a need for formal languages which describe biological systems similar to those used in physics or chemistry was highlighted in Nurse's 2008 paper [12]:

> "There should be a concerted programme . . . which will require both the development of the appropriate languages to describe information processing in biological systems and the generation of more effective methods to translate biochemical descriptions into the functioning of the logic circuits that underpin biological phenomena."
>
> (Paul Nurse (2008))

**Significance** AMIDOLseeks to provide a framework for the development of languages which formally describe biological systems, and complex systems from other domains, using an abstract intermediate representation which serves as a universal modeling representation. This framework provides several significant advancements on the state of the art which seeks to improve the usability, maintainability, reusability, and performability of scientific models. First, AMIDOL will provide compilers that translate from many Visual Domain Specific Ontological Languages (VDSOLs) into the abstract intermediate representation allowing VDSOLs to be specified for new domains and extended easily, and models written in VDSOLs to be transformed into a universal representation that preserves metadata about the original ontology. Second, AMIDOL will provide transformations on models in its abstract intermediate representation to optimize the execution of these models, compose models to build more powerful and complex representations, and to overlay reward structures on these models to specify metrics of interest and experiments to perform with AMIDOL's inference engine. Finally, AMIDOL provides compilers that translate models in the abstract intermediate representation into solver targets for the inference engine, checking constraints of composed reward structures, to solve for measures of interest. AMIDOLallows these

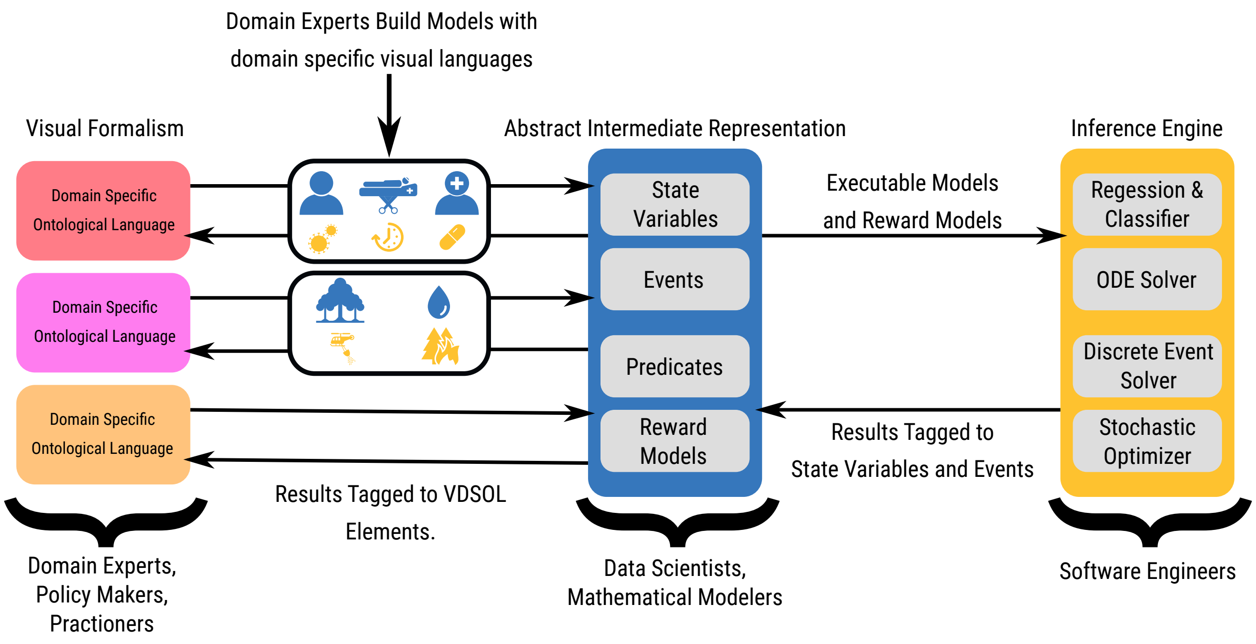

Domain Experts Build Models with
domain specific visual languages
Visual Formalism
Domain Specific Ontological Language
Domain Specific Ontological Language
Domain Specific Ontological Language
Abstract Intermediate Representation
State Variables
Events
Predicates
Reward Models
Executable Models and Reward Models
Inference Engine
Regession & Classifier
ODE Solver
Discrete Event Solver
Stochastic Optimizer
Results Tagged to State Variables and Events
Results Tagged to VDSOL Elements.
Domain Experts, Policy Makers, Practioners
Data Scientists, Mathematical Modelers
Software Engineers

Figure 2: AMIDOL Architecture

results to be communicated back to the user through the saved metadata, to relate raw statistical and machine learning output to the original ontology to improve explainability.

## 2 AMIDOL

Figure 2 outlines the architecture, and usage pattern for AMIDOL. AMIDOL presents a number of VDSOLs to the user,

### 2.1 Visual Domain Specific Languages

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.

## 2.2 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* [14, 17] and *event-synchronization* [**?**].

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.

## 2.3 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 [10], Generalized Stochastic Petri-nets with inhibitor arcs [4] which have been shown to be Turing complete, and stochastic activity networks [11, 16] 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.

## 2.4 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.
- $L : S|E \to \mathbb{N}$ is the event and state variable labeling function that maps elements of $S$ and $E$ into the original ontology.
- $\Phi : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \to \{0, 1\}$ is the event enabling predicate.
- $\Lambda : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \to (0, \infty)$ is the transition rate specification.
- $\Delta : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \to N_0 \times N_1 \times \ldots \times N_{n-1}$ 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.

## 3 Reward Variables, Reward Models, and Inference

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).

[13, 6, 5, 15]

### 3.1 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.

### 3.2 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.

### 3.3 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 $\theta$ 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 $\Theta_t$ 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 [9] 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.

## 4 Compartmental Model for Epidemiology

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 [8] and Ebola [7]. This model also lends itself to testing AMIDOL's Design of Experiments module via the plentiful CDC Data [3] 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.

## 5 Conclusions

## 6 Future Work

## 7 Acknowledgments

This research has been supported by DARPA contract DARPA-PA-18-02-AIE-FP-039.

## 8 Resources, web sites, etc.

MWS seeks to build a community and share resources, so feel free to have a section in your paper that points readers to web sites, github pages, etc.

## References

[1] Luca Cardelli. Brane calculi. In *International Conference on Computational Methods in Systems Biology*, pages 257–278. Springer, 2004.

[2] Luca Cardelli. Abstract machines of systems biology. In *Transactions on Computational Systems Biology III*, pages 145–168. Springer, 2005.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

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

[12] Paul Nurse. Life, logic and information. *Nature*, 454(7203):424, 2008.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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