Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

AGILE DESIGN MEETS HYBRID MODELS: USING MODULARITY TO ENHANCE HYBRID MODEL DESIGN AND USE

L. Kurt Kreuger
Weichen Qian
Nathaniel Osgood

Department of Computer Science
University of Saskatchewan
110 Science Place
Saskatoon, SK, S7N 5C9, CANADA

Kelvin Choi

Division of Intramural Research
National Institute on Minority Health
and Health Disparities
9000 Rockville Pike, Building 3, 5W05
Bethesda, MD 20892, USA

ABSTRACT

Dynamic modeling offers many benefits to understand the dynamics of complex systems. Hybrid modeling attempts to bring together the complementary benefits of differing dynamic modeling approaches, such as System Dynamics and Agent-based modeling, to bear on a single research question. We present here, by means of an example, a hybrid modeling technique that allows different modules to be specified separately from their implementation. This enables each module to be designed and constructed on an *ad-hoc* basis. This approach results in 3 benefits: it facilitates incremental development, a key focus in agile software design; it enhances the ability to test and learn from the behavior of a dynamic model; and it can help with clearer thinking about model structure, especially for those of a hybrid nature.

1 INTRODUCTION

There are many clear benefits to dynamic modeling. System Dynamics (SD) can efficiently capture certain key system patterns, such as feedbacks and accumulations (Homer and Hirsch 2006), can quickly describe aggregate characteristics of a system (enabling quick prototyping of models), and can generate insight into system behavior very effectively with causal loop diagrams and group model building techniques. They have been used in a wide range of fields, including many in health (Homer and Hirsch 2006), and public policy and business (Sterman 2000). Agent-based modeling (ABM), focusing on agent behaviors, allows one to pull apart the aggregate dynamics and describe in detail system dynamics, especially in the presence of certain characteristics, such as agent heterogeneity, social networks, and impacts of location (Rigotti and Wallace 2015). It has be used to study the formation of human norms (Dechesne, Di Tosto, Dignum, and Dignum 2013), noncommunicable diseases (Nianogo and Arah 2015), and even human migration patterns (Klabunde and Willekens 2016).

Clearly certain problems are more amenable to one modeling methodology than another. This can be due to several factors, such as different research goals (e.g. an initial foray into studying the patterns of a system might benefit most from an SD approach), the problem constraints themselves (e.g. if agent heterogeneity is known to be a key driving factor, ABM might be more beneficial), the nature of the particular stakeholder groups (e.g. whether they already use language and techniques from a system pattern, agent-behavior, or process perspective), or on the data that is available (aggregate data vs individual data).

A given problem might not have an obvious choice for modeling approach, however. The stakeholders might best understand the system pattern while the problem itself might have some need to describe agent-behavior. Here is where hybrid techniques might be implemented. Hybrid modeling attempts to bring to bear, simultaneously, the complementary benefits of multiple techniques, on a single research question.

Additionally, it is not uncommon for development paths to change. For example, if seeking to understand leverage points in a hospital emergency ward, an SD might be an efficient starting point to get an understanding of the main accumulations and feedbacks, and to assist in solidifying the types of questions sought by the model. As development progresses, it might become valuable to account for agent heterogeneity (e.g. nurses and doctors of different wards and specialties, or patients with various comorbidities), necessitating the use of an ABM. Taking a dynamic model as a thinking prosthesis, it is a distinct advantage to be able to not only switch between approaches model-wide, but also, through modular and incremental development, to alter different model components as assumptions and learning evolves.

Much work has been done on attempting to standardize hybrid techniques (Swinerd and McNaught 2012). The goal of this research, therefore, is to expand on the learning in hybridization techniques. We demonstrate a pattern for hybrid model construction that allows for easy connection between components of a model built with differing methodologies. This pattern, demonstrated with an example hybrid model, focuses on model modularity to support incremental development, and continual and efficient testing.

The benefits this technique offers are threefold. Firstly, proper modularization of a model supports incremental development, a key feature of the *agile* programming paradigm (Fowler and Highsmith 2001). This allows development to adapt more easily to changes in design requirements by enabling modules to be removed or added without requiring substantial changes to the model core. Given the role of models in driving insight that did not exist at the beginning of a modeling project, it can be seen how this might be a valuable feature of model design as well. Secondly, a modular design allows for improved testing and learning. Since modules are able to be simulated independently of each other, a given module's dynamics and response can be uniquely characterized. This improves investigation to see if unpredicted model behavior is due to fresh insight or code errors. Finally, for those software packages that allow multiple methods in the same single architecture (AnyLogic, being a prime example), modules themselves can be single-method even though the model as a whole draws on several, enabling clearer thinking about model structure. This could allow a unified method of model characterization building on current examples, such as the ODD method of ABMs (Grimm et al. 2006).

This paper is organized as follows: Section 2 outlines our technique of modularization and hybridization by means of an example model, including the discussion of 2 example cases used to demonstrate the technique's performance. Section 3 describes the scenarios that were run, and displays and compares the results. Section 4 discusses the insights gleaned from this work, including some current limitations.

2 OUR TECHNIQUE

The example model, built in AnyLogic (Version 7.1.2), describes the evolution of addiction to tobacco cigarettes in a human agent population. It uses components of ABM and SD to leverage their unique advantages. As shown diagrammatically in Figure 1, an ABM is used to describe the social network, and some agent characteristics (statecharts and variables). It further captures the physical environment (allowing a proximal network), and agent location (work or home). SD is encapsulated within the agent (in the *addictionModel* object), serving as the formalism of the addiction-related mental processes. It deals with much less quantifiable measures (e.g. level of addiction). This SD module, along with a Markovian version used to contrast behavior, will be described in the following sections.

The model consists of a number, n, of agents. All agents move continuously between a home (h in number) and a school (s in number), spending 16 hours at home and 8 hours at school (determined by simple timeout transitions). All agents are placed in a scale-free social network (described in the AnyLogic API as being generated according to the Barabási–Albert algorithm (Barabási and Albert 1999)). In this initial model, agents are not currently distinguished by demographic features, such as age or gender.

There can be multiple agents in each home and in each school. At model startup, agents are randomly assigned to a home. Schools and homes are distributed uniformly throughout a bounded 2D space. An agent is assigned a school based on the closest option from their home. Agents are aware of their social

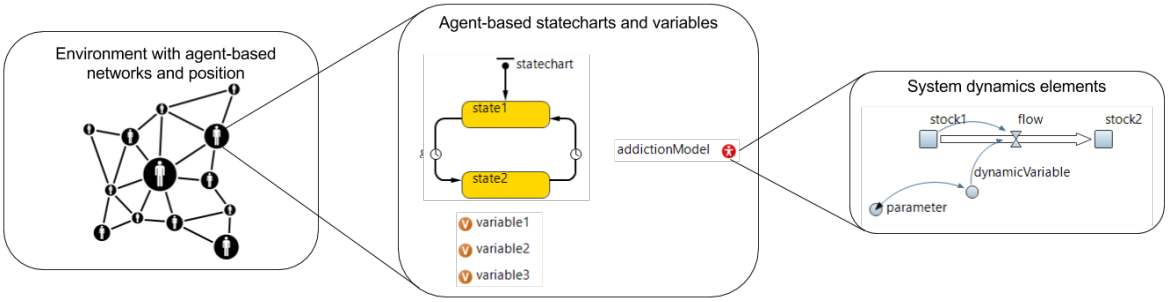


Figure 1: Our hybrid technique.

network (agents connected in the above-mentioned scale-free network), and their proximal network (agents occupying the same space, either home or school, at a given point in time).

Agents also can occupy one of 3 different states in a smoking statechart, depicted in Figure 2, *NeverSmoker*, *CurrentSmoker*, and *FormerSmoker*. *CurrentSmoker* agents will either be in the *NotSmoking* or *Smoking* state. Four of the five state transitions are rate transitions (i.e. *initation*, *quit*, *relapse*, and *haveACigarette*), whose transition times are sampled from an exponential distribution with the average transition time being specified by λ. The 5th state (i.e. *finishedCigarette*) is simply a timeout transition that corresponds to the length of time spent in the Smoking state, set in this model to be 10 model-minutes.

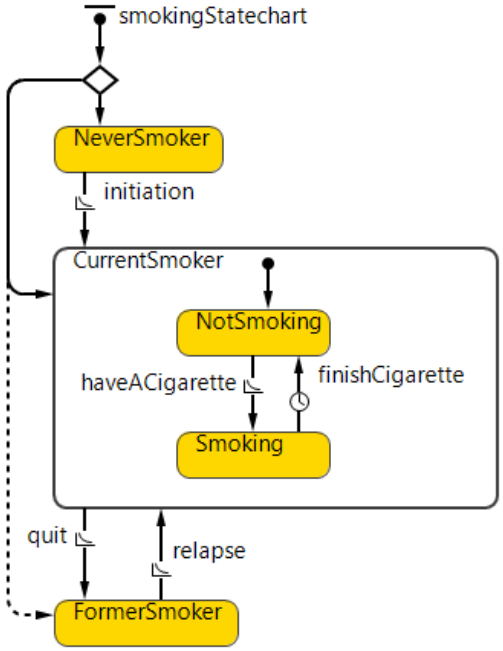


Figure 2: Our hybrid technique.

To define the time evolution of these 4 rate transitions, each agent is given an *AddictionModel* module, which encapsulates the necessary calculations. A Java Interface is used to define a contract that all individual implementations of the *AddictionModel* object must follow, enabling easy adaptability. The goal is to contain all the addiction-related logic, arguably one of the most uncertain components of this model, in a single module.

2.1 Modularity Through The Java Interface

Since AnyLogic is built on Java, we are able to leverage the Java Interface. An Interface is a contract that defines the main methods of any Java class that implements it. It formalizes relationships between different components of a software system, enhancing transparency. The Interface *IAddictionModel* defines the contract for this example: every *AddictionModel* module that implements the Interface will have at least 5 basic functions; 1 corresponding to each of the 4 rate transitions (taking no input and returning a *double*, the respective rate for that function), and 1 which takes as input the *Person* object, and outputs *void* (Java for "no output"). This input function provides the addiction module with access to all of the agent's parameters so, regardless of the design of the addiction module, it will have access to any parameters necessary for its normal functioning. The code is presented below in Figure 3. The next sections give examples.

```
public interface IAddictionModel
{
    public double getInitiationHazard();
    public double getSmokingHazard();
    public double getRelapseHazard();
    public double getQuittingHazard();
    public void updateModelValues(Person p);
}
```

Figure 3: *IAddictionModel* interface code.

Each agent has a setup parameter called *myAddictionModel*, which is of type *IAddictionModel*. Due to Java's polymorphic structure, this allows *myAddictionModel* to contain any addiction module that implements that Interface regardless of its internal structure. A setup function is defined which populates *myAddictionModel* with one of the already-defined addiction modules.

This setup function takes input from a global parameter, specifiable from an AnyLogic experiment. An experiment in AnyLogic is a particular encapsulation of a given simulation. Among other things, it defines the global parameters. Multiple experiments can be defined that specify different initial parameter setups. Therefore comparing multiple different addiction modules is as simple as changing an input parameter.

In order to replace a given addiction module with another that has been independently constructed (e.g. with stocks and flows, or ABM statecharts) and initialized (i.e. with defined initial parameter values), only 3 steps are needed to plug it in to the greater model. Firstly, any inputs this new module requires to function must be defined and calculated by the *Person* in which it is embedded. Secondly, the setup function needs to be expanded to include the new module as an option (which is very simple given the setup function is basically a switch statement). Finally, available modules are named in a Java *Enum* class, which is simply a list of options. A new element in this list must be specified with the name of the new module. Then, whenever an experiment is defined that uses this new module, it will run with it. This presents a very modest amount of extra work.

We now give 2 examples of different addiction modules that can be easily implemented and swapped.

2.2 Markov-inspired Addiction Module

To make predictions of tobacco prevalence, Killeen (2011) uses a 3 state Markov model (active smoking, recent cessation experiencing withdrawal, extended cessation), calibrating model parameters to various published research. This serves as an inspiration for our first addictionModel example. Our smoking statechart (Figure 2) is already essentially a Markov model, with slightly different states. As a result, each of the 4 output functions simply returns a daily probability for each transition required. This addiction module uses a daily probability for quitting and relapse that approximates some of the data reported in

(Killeen 2011). Triangular distributions (T{min, max, mode}) are used to generate daily probabilities. *getQuittingHazard()* returns a draw from T{0, 0.19, 0.095}, and *getRelapseHazard()* a draw from T{0, 0.162, 0.081}. Initiation, not accounted for in the original research, is assumed to be half as likely as relapse, so *getInitiationHazard()* is drawn from T{0, 0.081, 0.0405}. Finally, for current smokers, *getSmokingHazard()* returns a sample taken from T{0, 30, 10}, signifying that smokers will have up to 30 cigarettes per day, but averaging at around 10, or half a pack.

It should be mentioned that there are 2 main differences between this model and the usage by Kileen. Since Killeen is modeling the behavior of smokers exclusively, that project omits the *NeverSmoker* state, which we include. And while Killeen distinguishes between recent and long-term cessation, we do not.

This first addiction module does not contain the necessary logic to enable agents to change their addiction level over time based on external conditions. It is used as an example to demonstrate the flexibility of the modeling pattern employed here. However, it can be readily employed to validate other components of the model. By developing a simple standard, changes to network topology, for example, can be observed separately from the variations in agent decision making resulting from a more articulate addiction module.

2.3 Perceptual Control Theory Addiction Module

The second addiction module uses a drastically different set of assumptions. Relying on the ability of system dynamics to describe less quantifiable terms, and formalizing it using the Perceptual Control Theory (PCT) framework of (Powers 1973), we built a double-loop-learning structure (indicated in Figure 4). The format here is 2 hierarchical control feedback loops. Looking at the bottom action loop, a person has an *addictionLevel*, but what they perceive, *perceivedAddictionLevel*, is influenced by external signals (conceptualized here as the sum of *addictionLevel* and *externalSignals*). Their goal is to control the perception of their addiction, rather than their addiction itself. The value *reference1* is the set-point for this control loop. The person will then change their action (by means of the system dynamics flow) in order to bring the perceived addiction level in line with the reference point.

In this conceptualization, external signals correspond to observe smoking events – the average smoking rate of each network, either social (the scale-free network) or proximal (other agents in the same home or school). Essentially, a person's perceived addiction will be their actual addiction level reduced by how much smoking they see around themselves. If they smoke less often than their neighbors and peers, they will consider themselves less addicted than if they smoked the same amount while surrounded by non-smokers. The parameters *friendImpact* and *neighbourImpact* are multiplicative constants that determine the effect on an agent's perception of seeing a friend or neighbor smoke. They are each uniformly distributed on [-1, 0].

The upper layer characterizes the intent loop. This loop controls the set-point of the behavior loop, but otherwise has the same structure. The difference between the true addiction intent and the perceived intent is, again, the external signals. The addiction intent will evolve over time based on the discrepancies between their perceived intent and this loop's set point. This set point, *goalIntentState*, is a given agent's idealized picture of themselves – it is their idealized intent state. In this model, it is drawn from a uniform distribution on [0, 10], indicative of some individuals who desire to see themselves as smokers.

The parameters *abilityToChangeAddictionIntent* and *abilityToChangeAddictionLevel* multiply their respective error value and affect the speed that the respective stock is able to change value. They are 0.01, and 0.1, respectively, indicative of an ability to change one's physical addiction state more quickly than one's mental addiction state. Finally, *weightOfLevelOnGoal* is another multiplicative term that is simply set to 1 in this case. It is kept only for future generalizability.

In this second model, relapse is taken as the maximum of *addictionLevelIntent* and 0, since transition probabilities cannot be negative. We treat it as a proxy for how much an agent would like to be a smoker. Following the approach for the first addiction module, we treat initiation as half relapse, recognizing that relapse is easier than initiation. The transition to quit is taken as the maximum of -10 times *addictionLevelIntent* and 0. When *addictionLevelIntent* is negative, the agent has a negative desire to be a smoker, and so a

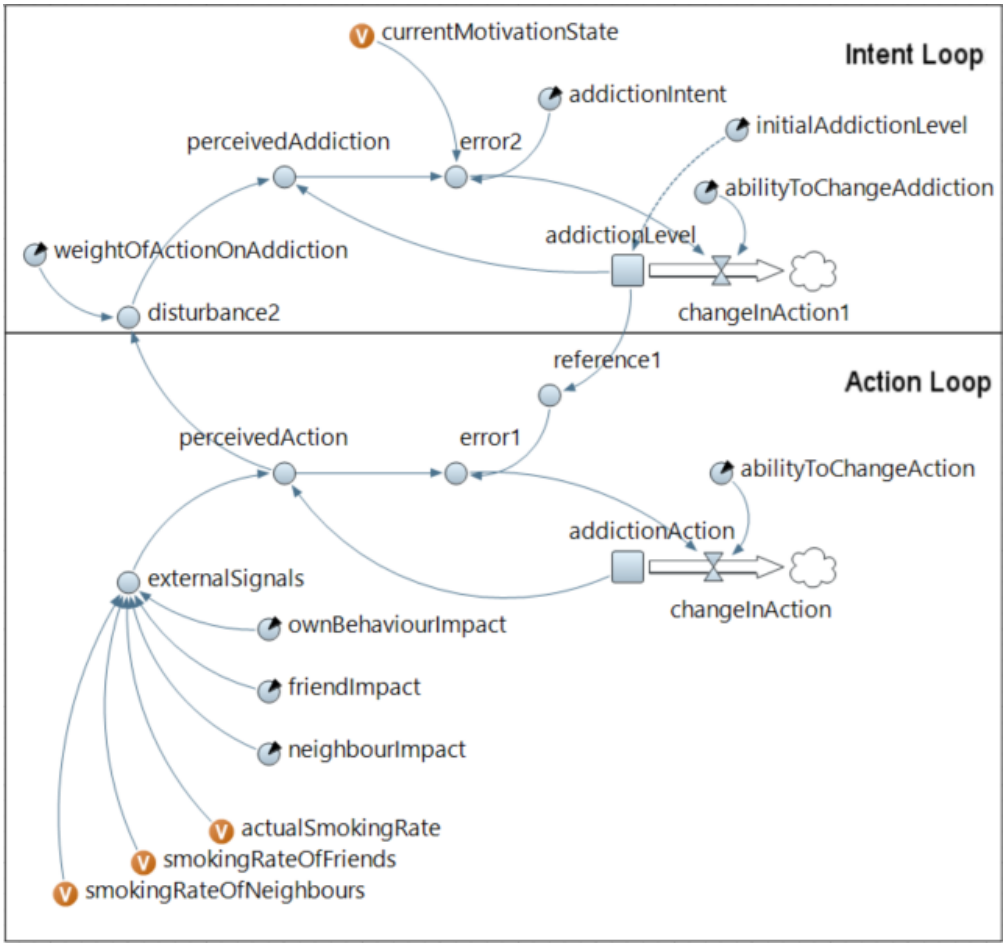


Figure 4: The Perceptual Control Theory addiction module.

positive desire to be a non-smoker. Finally, the *smoking* transition is determined from the *addictionLevel* stock, the measure of the strength of an agent's addiction.

3 RESULTS

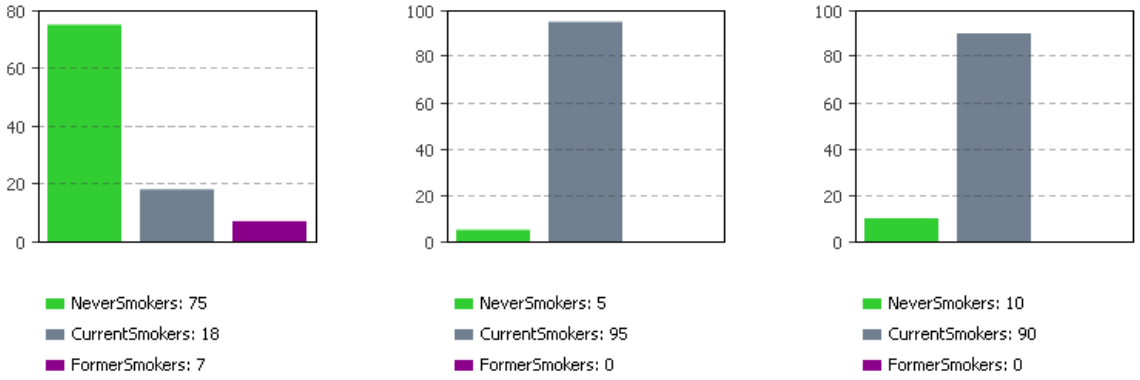
For each of the simulation runs conducted, the starting population was the same: 100 agents, 5 schools, 50 homes. Each person had a 20% chance of starting as a CurrentSmoker, and an 80% chance as a *NeverSmoker*. All agents were connected in a scale-free network (M = 10). For each run, 3 outputs are reported: the final proportions of the 3 different smoking statuses, the daily smoking rates of all current smokers at model end, and population-wide smoking-rate averages over the time of the model. These were chosen to show the different model outputs, both at model end and over time, of smoking status and use. All simulations were run for 1000 model days.

Given that the PCT model allows incorporation of network effects, we also ran this model with no network effects (i.e. by setting *friendImpact* and *neighbourImpact* to 0), so as to enable a better comparison with the no-network Markov version. Both network and non-network version of the PCT model were initialized with an $addictionLevel = \text{uniform}(0, 10)$, and $addictionAction = 0$. These initialization parameters can be, in general, quite important for the time-evolution of model parameters. We have not attempted to characterize this sensitivity, nor did we allow the stabilization of addiction stocks (through a warm-up period), in part because the dynamics themselves are part of the difference between the 2 addiction module approaches. Future research in using the PCT model will better characterize this behavior.

A word should be mentioned on the stochastic nature of these models. A proper analysis of output for such models would include a measure of the variability of the output given the same starting conditions. However, the focus of this research is on the nature of this particular hybrid design approach. The reported outputs are representative of a typical simulation run, and focus is on comparing the patterns that appear rather than the specific values.

3.1 Markov Addiction Model

Figure 5a shows the final smoking status distribution. With the Markov chain addiction module, the count of never smokers would stay relatively high, around 70 to 80. And current smokers usually outnumbers former smokers. This was a generally consistent pattern over multiple simulation runs. For smokers, the smoking frequency usually had a mean between 10 and 20 cigarettes per day, and was a two-tailed distribution, as seen in Figure 6a. Finally, from Figure 6b, the average smoking rate of the whole population (including non and former smokers) did not display much variation, and stayed relatively consistent between 2 and 4 cigarettes per day.



(a) Markov. (b) PCT with network effects. (c) PCT without network effects.

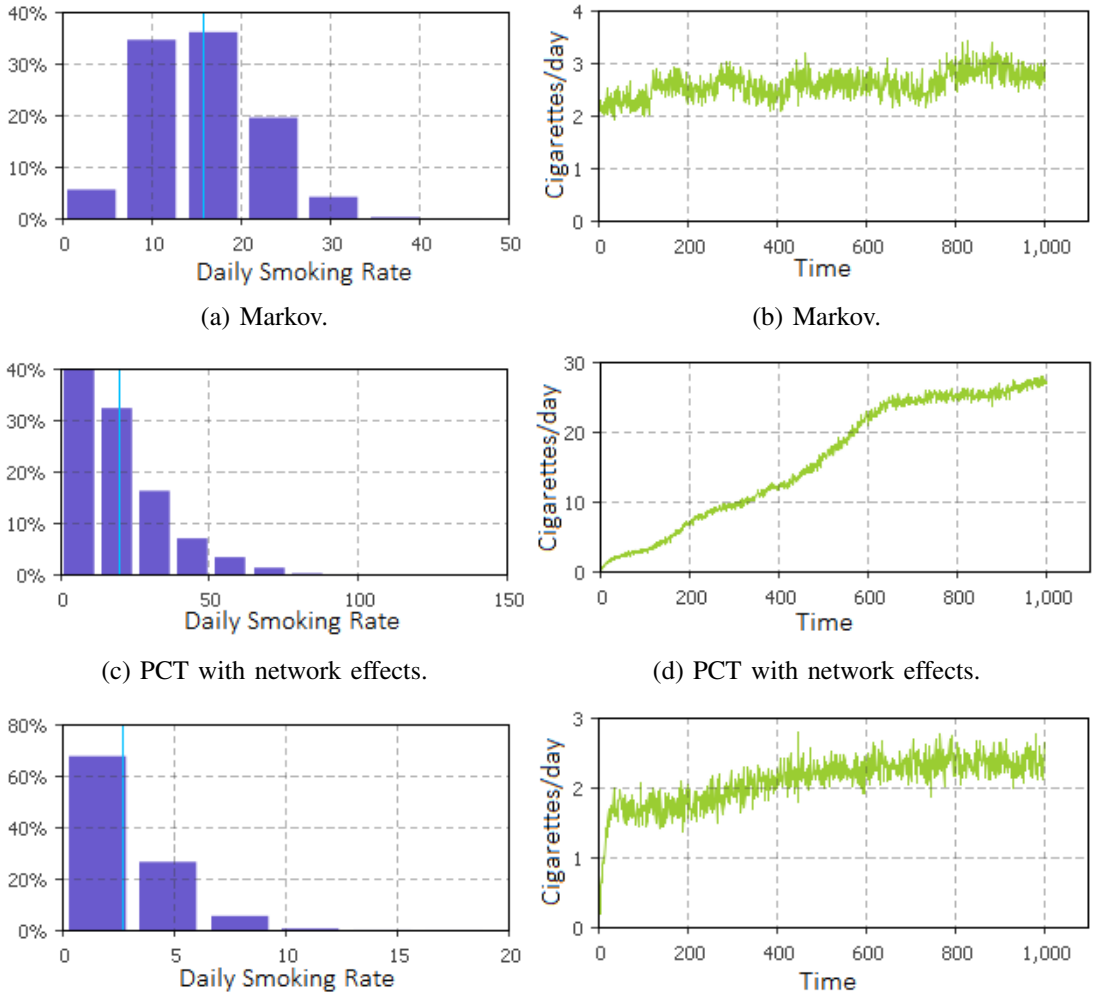
Figure 5: Histogram of final smoking status.

3.2 PCT Addiction Model

We can see from Figures 5b and 5c that with the SD addiction module, nearly the whole population became and remained active smokers, whether networks were used or not. This, obviously, is not realistic for most populations, but displays a dramatic departure from the Markov addiction module.

Figure 6c shows notable differences in the smoking rates for active smokers. Rather than a two-tailed distribution, most agents here smoke relatively few cigarettes, with a single tail reaching much farther up to more than 3 packs per day. Without a social network, Figure 6e, the single tail structure remains with much smaller smoking rates among smokers (most under 3 cigarettes per day). There is a large apparent effect of the social network on smoking intensity, but less so on the shape of the distribution.

Finally, Figure 6d shows a gradual increase in population smoking rate, likely due mostly to the increase in smoker initiation. Reinforcing this point, removing network effects (Figure 6f) shows a largely flat population smoking rate curve. While initiation is not caused by the social network, it is clearly accelerated by it.



(a) Markov.

(b) Markov.

(c) PCT with network effects.

(d) PCT with network effects.

(e) PCT without network effects.

(f) PCT without network effects.

Figure 6: Histogram of daily smoking rates for smokers and population-wide average of daily smoking rates over time.

4 DISCUSSION

Clearly the two addiction modules make vastly different assumptions. One uses unchanging, individual parameters and neglects network or internal feedback effects. The other relies on network and feedbacks, and incorporates a framework for individual addiction dynamics. The Markov model outputs consumption plots with 2 tails and a clear central maximum. The PCT method predicts a single tail decline. Further, since the PCT method takes into account social network, the outputs are drastically different.

While neither of the demonstrated addiction modules are validated to real-world data, they serve as a very useful test case to display the benefits of this modeling pattern. As mentioned in the introduction, one benefit is the ability to adapt to changing model requirements through an incremental approach, an important part of agile software development. If initial requirements assume that agent heterogeneity and location are key interests to examine in the model, the Markov module might be a good choice. If throughout the modeling process it becomes clear that agent behavior needs to be more complex, without resorting to building a new model or substantial recoding of core model behavior, a more complex PCT module can

be implemented. The model design is relatively ambivalent to which modeling approach is used, allowing many possible hybrid combinations.

There is a need for comparative testing of behavior theories. This is currently done with human subject research, which is costly and time-consuming, e.g. (Sutton, McVey, and Glanz 1999), (Elliott and Ainsworth 2012). Using this basic architecture, it is quite easy to imagine the development of any number of differing addiction modules based on, for example, the Theory of Planned Behavior or the Health Belief Model, and using any combination of aggregate or individual modeling techniques. The ABM framework is indifferent to the technique, so long as the contract is matched. The many benefits of dynamic simulation can be extended to this area of comparative research.

Increased learning and testability was also mentioned. Since the module has been reified as a model parameter itself, it can be treated as such in sensitivity analyses. Here, it is clear that there are substantial differences in output between modules, indicating that it is a highly sensitive parameter. This could suggest more work go into the internal mental model rather than other features (e.g. GIS). And, module behavior can be tested with very simple agent populations, allowing selective and precise testing plans.

Finally, this pattern has specific benefits to hybrid models for those packages that allow hybrid techniques (including purely code-based approaches). Formalizing a set of design patterns has helped standardize software engineering practices (Hohpe et al. 2013). This pattern, then, can serve as another step in the same process for hybrid dynamics models.

Compartmentalizing each technique into a distinct module allows for extensions unique to that approach. For an SD model, due to an often aggregated approach to system parameters, adding a new causal pathway (for example, adding the influence of law makers on tobacco prices) might be a relatively straightforward addition, whereas for an ABM this might add substantial extra mechanism (for example, requiring the introduction of law maker behavior under a variety of model conditions). ABMs, on the other hand, can easily add new agent-level characteristics (for example, including a new demographic), which would require complex subscripting in an SD model. Separating the details of module implementation through an abstraction of the Java interface allows each module to be extended relatively freely from each other.

Hybrid modeling is not without challenges, however. While SD models benefit from no increased time complexity with increasing population size, ABMs are required to track each agents behavior and therefore do often take more computational resources. Hybrid modeling can have worse scaling that pure ABMs. This is at least partly due to the current implementation of stocks and flows in AnyLogic. Through in-house trials, we have observed that with many events or state transitions, all flows model-wide, of which each agent has several, are recalculated, leading to a drastic slow-down when compared with pure ABM implementations. Even this simple model would not efficiently run with a population of even tens of thousands (taking nearly 1 day for 10,000 agents using the PCT addiction module on a Core i7-4700HQ laptop). However, this serves as a further example of the benefits of this modeling technique: by modularizing the design, an SD module could be constructed that captures the concepts (leveraging SD's ability to facilitate conceptual mapping), and when scaling to larger populations an ABM version could be implemented. The behavior of each module can be readily compared ensuring accurate translation had occurred.

This pattern further assumes that an individual-based framework serves as the substrate. This do not generalize to all possible hybrid techniques. It will be up to future work to examine if this pattern can be extended.

A particular weakness to the example model here described is that, since it is built in the software package AnyLogic, the implementation is unique to that software package. However, it more generally relies on the Java Interface, and so could be readily implemented with only a few changes in Repast or in a pure Java framework. It would require some research to develop an implementation in a non-Java framework.

REFERENCES

Barabási, A.-L., and R. Albert. 1999. "Emergence of Scaling in Random Networks". *Science* 286 (5439): 509–512.

Dechesne, F., G. Di Tosto, V. Dignum, and F. Dignum. 2013. "No Smoking Here: Values, Norms and Culture in Multi-Agent Systems". *Artificial Intelligence and Law* 21 (1): 79–107.

Elliott, M. A., and K. Ainsworth. 2012. "Predicting University Undergraduates' Binge-Drinking Behavior: A Comparative Test of the One- and Two-Component Theories of Planned Behavior". *Addictive Behaviors* 37 (1): 92–101.

Fowler, M., and J. Highsmith. 2001. "The Agile Manifesto". *Software Development* 9 (8): 28–35.

Grimm, V., U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Heinz, G. Huse et al. 2006. "A Standard Protocol for Describing Individual-Based and Agent-Based Models". *Ecological Modelling* 198 (1): 115–126.

Hohpe, G., R. Wirfs-Brock, J. W. Yoder, and O. Zimmermann. 2013. "Twenty Years of Patterns' Impact". *IEEE Software* (6): 88.

Homer, J. B., and G. B. Hirsch. 2006. "System Dynamics Modeling for Public Health: Background and Opportunities". *American Journal of Public Health* 96 (3): 452–458.

Killeen, P. R. 2011. "Markov Model of Smoking Cessation". *Proceedings of the National Academy of Sciences* 108 (Supplement 3): 15549–15556.

Klabunde, A., and F. Willekens. 2016. "Decision-Making in Agent-Based Models of Migration: State of the Art and Challenges". *European Journal of Population* 32 (1): 73–97.

Nianogo, R. A., and O. A. Arah. 2015. "Agent-Based Modeling of Noncommunicable Diseases: A Systematic Review". *American Journal of Public Health* 105 (3): e20–e31.

Powers, W. T. 1973. *Behavior: The Control of Perception*. Aldine Chicago.

Rigotti, N. A., and R. B. Wallace. 2015. "Using Agent-Based Models to Address 'Wicked Problems' Like Tobacco Use: A Report From the Institute of Medicine". *Annals of Internal Medicine* 163 (6): 469–471.

Sterman, J. D. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World*, Volume 19. Irwin/McGraw-Hill Boston.

Sutton, S., D. McVey, and A. Glanz. 1999. "A Comparative Test of the Theory of Reasoned Action and the Theory of Planned Behavior in the Prediction of Condom Use Intentions in a National Sample of English Young People.". *Health Psychology* 18 (1): 72.

Swinerd, C., and K. R. McNaught. 2012. "Design Classes for Hybrid Simulations Involving Agent-Based and System Dynamics Models". *Simulation Modelling Practice and Theory* 25:118–133.

AUTHOR BIOGRAPHIES

L. KURT KREUGER is a Ph.D. student at the University of Saskatchewan. He holds a Master's in Physics and Engineering Physics, a Bachelor's in Computer Science, and a Bachelor's in Engineering Physics. His research interests lie in behavioral modeling, and integrating dynamic modeling and machine learning. His email address is kurt.kreuger@usask.ca.

KELVIN CHOI is the Stadtman Investigator and the Acting Head of the Social and Behavioral Group at the Division of Intramural Research of the National Institute on Minority Health and Health Disparities. He holds a Ph.D. in Epidemiology, an M.P.H. in Community Health Education, and a B.Sc. in Physiotherapy. His research focuses on social and behavioral epidemiology, particularly in tobacco use disparities and tobacco control, and using computational models to simulate potential population impact of tobacco control strategies. His email address is kelvin.choi@nih.gov.

WEICHANG QIAN is a Ph.D. student at the University of Saskatchewan. He holds a Master's in Computer Science and a Bachelor's in Industrial Engineering. His research interests lie in applying simulation model-

ing with machine learning on sensor data of human behavior. His email address is weicheng.qian@usask.ca.

NATHANIEL OSGOOD is Associate Professor in the Department of Computer Science, Associate Faculty in the Department of Community Health and Epidemiology, and Director of the Computational Epidemiology and Public Health Informatics Laboratory. His email address is nathaniel.osgood@usask.ca.