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.

POPULATION-BASED CTMCS AND AGENT-BASED MODELS

Tom Warnke
Oliver Reinhardt
Adelinde M. Uhrmacher

Institute of Computer Science
University of Rostock
Albert-Einstein-Str. 22
D-18059 Rostock, GERMANY

ABSTRACT

Currently, only few agent-based models are implemented with a continuous representation of time, although state-of-the-art agent-based modeling and simulation (ABMS) frameworks support continuous-time models and continuous time often allows for a more faithful capturing of reality. Intrigued by this discrepancy, we take a closer look at population-based Continuous-Time Markov Chains (CTMCs), their modeling and their simulation, on the one hand, and, how continuous-time agent-based models are currently realized in state-of-the-art ABMS frameworks, like Repast Simphony and Netlogo, on the other hand. Subsequently, we adopt and adapt concepts and algorithms of modeling and simulating population-based CTMCs. We propose a solution how to integrate those into contemporary ABMS frameworks which results in a more succinct description of continuous-time agent-based models.

1 INTRODUCTION

Agent-based modeling and simulation (ABMS) is a well-established scientific method. A number of software packages exist that support scientists in developing and using agent-based models (Railsback, Lytinen, and Jackson 2006). Most notable examples are Repast Simphony (North et al. 2013), MASON (Luke et al. 2005), and NetLogo (Wilensky 1999). All these simulation tools offer a simple way to schedule events repeatedly at equidistant time points. Thus, the simulation paradigm of *discrete time step simulation*, where the system changes its state at fixed points in time, can be readily implemented. In Repast and MASON, and in NetLogo (using an extension (Sheppard and Railsback 2015)), events can also be manually scheduled at arbitrary points in continuous time, allowing for *discrete event simulation*.

However, reviews of scientific applications of these ABMS frameworks frequently find that the vast majority of models is designed with a step-wise schedule (Railsback, Lytinen, and Jackson 2006). Recent examples for influential step-wise agent-based models from the social sciences include the "Wedding Doughnut" model (Silverman et al. 2014) implemented in Repast Simphony and the MASON RebeLand model (Cioffi-Revilla and Rouleau 2010). This observation is in stark contrast with inclinations in application communities to use models with continuous time-bases. For example, in demography continuous-time models are considered superior to discrete-time models, as they allow for higher precision and the integration of established domain-specific analysis methods (Willekens 2009).

Comparing the step-wise and the event-based approaches, a possible explanation for this divergence might be the easier mapping of available data to a step-wise model. For example, a typical demographic data set might contain information about which percentage of each age cohort in a population dies each year. In a simulation with a step size of one year, the probability of each agent to die in a step can be directly obtained from this data, given its age. The same applies to other demographic events such as

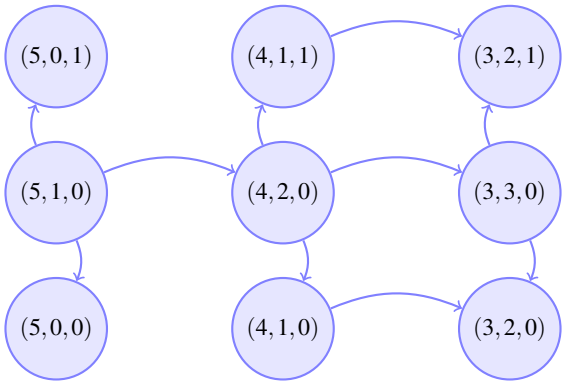

Figure 1: Snippet of the state space and the transitions in a population-based SIR CTMC model. The tuple in each state shows the size of the susceptible, infectious and recovered population, respectively. State transitions modeling an individual getting infectious (left to right), an individual recovering (upwards), and an infectious individual dying (downwards) are denoted.

marriage and childbirth. However, such a step-wise approach effectively models the available data. To model the underlying, data-generating processes, which are determined by individual decisions and events that do not happen equidistantly in time, a discrete event approach is more adequate (Willekens 2009).

In this context the waiting time, i.e., the time in between events, is of central interest. Often these waiting times are drawn from stochastic distributions. One particular class of models based on stochastic waiting times is the class of Continuous-Time Markov Chains (CTMC). CTMCs are a well-established method in many application domains of simulation, especially for population models (Henzinger, Jobstmann, and Wolf 2011). Consequently, a plethora of modeling formalisms, simulation algorithms, and scientific applications that employ CTMC population models exists. For instance, CTMCs are the formal foundation of some rule-base modeling languages in systems biology (John et al. 2011, Warnke et al. 2015). Another example for modeling population dynamics with CTMCs are SIR epidemic models, where individuals change between susceptible, infectious, and recovered sub-populations (Allen and Lahodny 2012). Other phenomena, e.g., decision processes, can be modeled by competing stochastic waiting times as well (Warnke et al. 2015, Klabunde et al. 2015). CTMC models also provide possibilities for formal analysis, e.g., of performance and correctness (Baier et al. 2010), or by approximation with a set of ODEs (Ciocchetta et al. 2009).

In this paper, we integrate some methods that were developed for population-based CTMC models into ABMS. Our contribution is threefold. First, we want to encourage continuous-time agent-based modeling by demonstrating how appropriate metaphors and formalisms can support the modeler. Second, we show how effective discrete event simulation algorithms for agent-based models can be derived from CTMC simulation algorithms. Third, we propose an integration of these concepts in contemporary ABMS frameworks.

2 POPULATION-BASED CTMC AND HETEROGENEITY OF POPULATIONS

The fundamental assumption of population-based models is that the modeled individuals can be divided into a number of homogeneous sub-populations. From the model's perspective, the individuals in each population are completely equivalent. A state of the model can then be described by enumerating all populations. For example, each state (S,I,R) of a simple SIR model consists of three positive integers that represent the number of individuals being susceptible (S), infectious (I), and recovered (R). The model dynamics can be described by a state transition system. This possibly infinite transition system is a Continuous-Time Markov Chain if the waiting time distributions for transitions originating from a state s are exponentially distributed and depend only on s.

One transition in a CTMC SIR model represents a change of one individual, for example one susceptible individual becoming infectious (Allen 2008). Thus, one possible transition could be $(4,2,0) \xrightarrow{4\times2\times a} (3,3,0)$, meaning that, if 4 susceptible, 2 infectious and no recovered individuals exist, one possible successive state will be 3 susceptible, 3 infectious and no recovered. Obviously, the propensity of this transition needs to take the number of possible encounters between individuals of both sub-populations into account. Thus, if we assume that a is the rate constant with which a susceptible person falls ill if meeting an infectious person, the exit rate of the state will be $4 \times 2 \times a$ to transition to the state $(3,3,0)$. The average waiting time until one of the susceptible individuals is infected is $(4 \times 2 \times a)^{-1}$. If at the same time the rate for an infectious person to recover will be b, then the rate of this transition in the state $(4,2,0)$ is $2 \times b$: $(4,2,0) \xrightarrow{2\times b} (4,1,1)$. Both transitions originate in the same state, but lead to different successor states. Which one is actually executed is governed by their waiting time distributions. A higher rate corresponds to shorter waiting times and, thus, a higher probability of execution. This concurrency among transitions is called "stochastic race", as all transitions that are enabled in a state compete and the fastest one succeeds. The use of population sizes as rate factors is a common pattern in population models.

To add more detail to the individuals in the model, more sub-populations need to be distinguished. For example, if individuals shall also be distinguished by sex, each model state must contain 6 populations (SF, IF, RF, SM, IM, RM). To also differentiate between age cohorts, e.g., 10 different ones, each state must already contain 60 sub-populations. Obviously, the number of populations to model explodes when the number of attributes increases. Also, we have only looked into attributes with discrete domains yet. If individuals are equipped with at least one continuous attribute, the set of populations (i.e., a single model state) is infinite. Assuming that agents in agent-based models are highly heterogeneous with a larger number of attributes than in the previous example, a categorization into populations of equivalent agents is futile.

As CTMCs are potentially infinite, modeling formalisms are required for a finite specification. Many modeling formalisms have been developed that allow a more or less succinct description of CTMCs (Henzinger, Jobstmann, and Wolf 2011). E.g., $s + i \xrightarrow{\#s\times\#i\times a} i + i$ defines the event of infection as a rule that replaces one susceptible and one infectious agent with two infectious agents, with a rate based on the number of susceptible and infectious agents in a population and a constant. This rule captures all transitions from left to right in Figure 1. Although rules are only one way to describe CTMCs, the important property of CTMC modeling formalisms becomes evident: each rule can describe an infinite number of state transitions by defining a state transition function. The state transition function can be applied to each of an infinite number of states to obtain a successor state. Models with CTMC semantics, e.g., defined in the rule-based modeling language ML-Rules (Warnke, Helms, and Uhrmacher 2015), can be executed with stochastic simulation algorithms, which are detailed in the next section.

3 STOCHASTIC SIMULATION ALGORITHMS

Examples for stochastic simulation algorithms (SSAs) to execute models with CTMC semantics are the so called Doob-Gillespie algorithms (Doob 1945, Gillespie 1976, Gillespie 1977). These algorithms sample a trajectory from the CTMC defined by the model by repeating a simple step. Given the current simulation time t_i and the current model state s_i the algorithms determine the time of the next state change t_{i+1} and the next state s_{i+1} following the CTMC's distribution. The initial state s_0 of the model at time t_0 is given by an initialization. The variations of the stochastic simulation algorithm differ in the implementation of this step.

One of the basic SSA variants originally introduced by Gillespie is the First Reaction Method. In the First Reaction Method the basic SSA-step described above consists of three parts:

1. Sample a waiting time Δt_{i+1}^k for each transition k that is possible in the current state s_i from the transition's waiting time distribution. As waiting times follow an exponential distribution with

parameter $\lambda_k(s_i)$ this can be efficiently done using the inversion method:

$$\Delta t_{i+1}^k = \frac{1}{\lambda_k(s_i)} \ln \frac{1}{r_k}, \qquad r_k \sim \text{uniform}(0,1) \tag{1}$$

2. Find the transition μ with minimal Δt_{i+1}^k.
3. Set t_{i+1} as $t_i + \Delta t_{i+1}^\mu$ and determine s_{i+1} by applying the transition μ.

The First Reaction Method directly implements the idea of the stochastic race. All possible transitions compete against each other and the one with the shortest waiting time drawn wins. Transitions with higher transition rates $\lambda_k(s_i)$ are chosen with higher probability.

These algorithms work well for models with only a few competing transitions in each state, such as the chemical reaction networks they were originally designed for or the previously mentioned population-based SIR models. However, the more transitions have to be considered, the less efficient the SSA becomes, as the algorithm draws a random number for each possible transition in every state, among which only one will be used. In agent-based models the amount of possible transitions given one state usually is very large. For example, in the described population-based SIR model only one state transition exists that corresponds to a person becoming infected, independent of the size of the susceptible sub-population (the size only affects the rate of the transition), as different individuals in the population can not be distinguished.

In an agent-based model there would be a large number of transitions as the infection of distinguishable agents would result in different follow-up states. Populations of agents tend to be heterogeneous, thus rendering a direct implementation of stochastic race in larger agent-based models infeasible: at each simulation step a high number of waiting times will be calculated never to be used.

The above algorithms recalculate the waiting time for all possible transitions and then select the fastest, which is possible due to the Markov property of the exponential distribution. An alternative are algorithms that only recalculate waiting times on demand. These algorithms rely on event queues to store future transitions and the time they take place. Event queues are priority queues (Brown 1988), in which the time stamp of an event is used as its priority. Classic event queues support two main operations: enqueue, to insert a new event in the queue, and dequeue, to extract the event with the smallest time stamp to be executed.

In addition to enqueue and dequeue events, sometimes the time that events are scheduled needs to be updated. For example, in a SIR model for an infectious agent recovery and death may be scheduled as future events. If the recovery happens first, the scheduled death event of the now healthy agent might be rescheduled at a later time point. Thus, the simulation algorithms will rely on a further operation on queues, i.e., the requeue operation. One example of simulation algorithms that entirely rely on the requeue operation are algorithms for models of the DEVS family (Zeigler, Praehofer, and Kim 2000). With each model component a time of next event is associated. Therefore, the number of events scheduled overall remains constant. However, there also exist DEVS variants that supports dynamic structures and thus the generation or removal of model components, e.g., Steiniger and Uhrmacher (2016). Here, enqueue and dequeue are needed again to allow model components to be added and removed, in addition to the requeue operation (to determine the time of next event for a model component).

4 CONTINUOUS-TIME AGENT MODELS IN ABMS FRAMEWORKS

Our considerations in the previous section show that scheduling algorithms rather than SSAs are appropriate to implement agent-based CTMC models. We will now investigate how such a model can be implemented in current ABMS frameworks, and how their scheduling mechanisms support continuous-time modeling. Specifically, we consider Repast Simphony, MASON and NetLogo.

All reviewed ABMS frameworks directly support scheduling events at fixed time steps and, thus, discrete time step simulation. Repast Simphony and MASON, being based on the object-oriented programming language Java, incorporate an explicit schedule object in their simulation scheme. These schedule objects

```
1  public class Agent {
2
3    /* ... */
4
5    private ISchedulableAction scheduledEvent;
6
7    public void getInfected() {
8      this.infectionState = InfectionState.INFECTIOUS;
9      scheduleRecovery();
10     informNeighbours();
11   }
12
13   private void informNeighbours() {
14     for (Agent agent : network.getAdjacent(this)) {
15       agent.rescheduleInfectionEventIfPresent();
16     }
17   }
18
19   public void rescheduleInfectionEventIfPresent() {
20     if (infectionState == InfectionState.SUSCEPTIBLE) {
21       if(scheduledEvent != null) {
22         schedule.removeAction(scheduledEvent);
23       }
24       scheduleInfection();
25     }
26   }
27
28   private void scheduleInfection() {
29     double currentTime = schedule.getTickCount();
30     double infectiousNeighbors = getInfectiousNeighbors();
31     if (infectiousNeighbors == 0.0) {
32       scheduledEvent = null;
33     } else {
34       double rate = infectionRate * infectiousNeighbors;
35       double waitingTime = RandomHelper.createExponential(rate).nextDouble();
36       scheduledEvent =
             schedule.schedule(ScheduleParameters.createOneTime(currentTime +
             waitingTime), this, "getInfected");
37     }
38   }
39 }
```

Figure 2: Model code snippet from a Repast implementation of an agent-based SIR model. Each agent keeps track of the event it scheduled for itself (line 5). When an agent is infected, the method `informNeighbours` is called (line 10). This triggers all social contacts of the newly infectious agent (line 14) to retract their infection event (line 22), calculate the new rate (line 34) and draw a new waiting time (line 35). A new infection event is scheduled after the new waiting time (line 36).

provide an interface to schedule method calls on agents once or repeatedly with a fixed interval. Repast Simphony supports the use of Java annotations as well. The default behavior in NetLogo is to execute a central method at every time step, which usually contains calls to the agents.

Little work exists on the implementation of continuous-time models and discrete event simulation in ABMS frameworks, though. A tutorial for Repast Simphony shows how to implement a queuing system with exponentially distributed arrival and service times (Sweda 2011). An extension for NetLogo is available

```
1  to schedule-infection
2      let infected link-neighbors with [health = "infected"]
3      set infect-id (infect-id + 1)
4      if any? infected [
5        let waiting-time random-exponential 1 / (infection-rate * count infected)
6        let i infect-id
7        time:schedule-event self task [infect i] time:plus current-time waiting-time
             "day"
8      ]
9  end
10
11 to infect [i]
12   if i = infect-id [
13     set color red
14     set health "infected"
15     update-plot
16
17     ask link-neighbors with [health = "susceptible"] [
18       schedule-infection
19     ]
20     schedule-recovery
21   ]
22 end
```

Figure 3: Model code snippet from a NetLogo implementation of an agent-based SIR model. When an agent is infected (line 11), all social contacts that are susceptible reschedule their infection (line 18). They recalculate the infection rate and draw a new waiting time (line 5). As the NetLogo extension does not support the retraction of events, it has to be ensured that only the most recently scheduled infection event gets executed. Therefore, the event gets a unique id (lines 3 and 6) that will be compared with the last used id before the execution of the event (line 12).

that enables scheduling events in continuous time (Sheppard and Railsback 2015). In any case, the modeler is required to explicitly schedule events at a certain time point. To illustrate how agent-based models employ this approach, we will now sketch the implementation of a continuous-time agent-based SIR model in modern ABMS frameworks. The model is related to the discrete-time BioWar model (Carley et al. 2006), but much simpler.

Our model consists of a number of agents with their infection state as their only attribute. Similar as in the population-based model, the infection state can be either susceptible, infectious, or recovered. The agents are connected in a fixed network of bidirectional links (social contacts). Initially, a proportion of the agents is infectious, and agents are randomly linked. If an agent is susceptible, he becomes infectious after a random waiting time that is exponentially distributed with the rate $\#infectious \times a$, where $\#infectious$ is the number of infectious social contacts, and a is a rate constant. Thus, for an agent with no infectious contacts this rate will be zero and he will never become infected. If an agent is infectious, he recovers after a random waiting time that is exponentially distributed with the constant rate b. The parameters of the model are the total number of agents, the number of initially infectious agents, the parameters for the network construction and the rate constants a and b.

The risk of a susceptible agent to be infected depends on the infectious agents among its social contacts. For each susceptible agent this number is determined, the result is used to draw a waiting time, and an infection event is scheduled. Also, for each infectious agent a recovery event is scheduled. If now the first event is executed, a susceptible agent gets infected or an infectious agent recovers. All social contacts of the changed agent now have one more or one less infectious social contact. Thus, if they have scheduled an infection event, its waiting time distribution has changed. New waiting times are drawn and their infection

events are rescheduled. All other agents' infection events as well as all recovery events are not affected and remain in the event queue.

The simulation scheme as described above requires that events are manually scheduled and retracted. The snippets in Figure 2 and 3 demonstrate how this can be achieved using Repast Simphony and NetLogo. The complete models for NetLogo and Repast Simphony are available on-line at git.informatik.uni-rostock.de/mosi/netlogo-sir-model and git.informatik.uni-rostock.de/mosi/repast-sir-model.

5 LESSONS LEARNED IN THE CTMC WORLD

In the previous section we have shown how the modeler can exploit her knowledge about the locality of events to keep changes to the existing schedule as small as possible, so to allow for an efficient execution. This is in line with the dominant paradigm in agent-based modeling and simulation to implement advanced concepts by programming them explicitly into the model. This requires additional effort by the modeler and clutters the model with execution related details. There exists no clear separation between the model, i.e., a representation of the system of interest, and the simulation algorithm, i.e., a method to create data from models. However, such a separation of concerns has been identified as advantageous in software design in general and also in modeling and simulation (Himmelspach and Uhrmacher 2007, Wainer 2009, Sarjoughian, Zeigler, and Hall 2001). A very obvious advantage for modeling and simulation is that reusing model and algorithm independently becomes possible: the same model can be executed with different simulation algorithms, which can even be exchanged dynamically on demand to improve performance (Helms et al. 2015).

To emulate the positive effect of the modeler's knowledge on simulation performance, the algorithm must automatically exploit the locality of events. More specifically, after executing an event, it must automatically determine which already scheduled events have been affected by the recent change of the model state, reevaluate their rate expression, and reschedule them at a new time. This corresponds to the Next Reaction Method, another influential, more recent (in comparison to the original ones) SSA variant (Gibson and Bruck 2000). Here, a dependency graph is used to define which events are affected by the execution of one event. Only those scheduled events that are actually affected are rescheduled. For example, if an event decreases the number of recovered agents in the population by one, a scheduled infection event is unaffected. However, this event needs to be rescheduled in case infected or susceptible agents have died. Among other declarative modeling approaches, rule-based modeling languages allow for automatically determining interdependencies on a syntactical level by comparing the names of entities involved in rules.

For models with a finite set of rule instantiations (or analogous descriptions of state transitions) the dependency graph can be computed once before the model is executed (Versari and Busi 2008). However, if the modeling formalism allows for an infinite number of rule instantiations, for example through dynamic hierarchies (Warnke, Helms, and Uhrmacher 2015), the dependency graph is infinite as well and can not be computed in advance. Approaches that only calculate dependencies between currently active rules exist as well, e.g., the NFsim algorithm (Sneddon, Faeder, and Emonet 2011).

An approach that is closely related to the Next Reaction Method aims at the stochastic simulation of reaction-diffusion systems in biology (Elf and Ehrenberg 2004). In the Next Subvolume Method space is partitioned into subvolumes, i.e., spatial areas that contain populations of entities. Two kinds of events can occur: a reaction of entities inside one subvolume or a diffusion of an entity from one subvolume into a neighboring one. In the first case the populations in only one subvolume are affected by the event, in the second case the populations in two subvolumes are affected. Thus, only events that are scheduled to occur in these affected subvolumes have to be reconsidered, and events in all other subvolumes remain unchanged. The key to apply such an optimization again lies in determining which events are affected. In the Next Subvolume Method this dependency is given by the spatial context in which a reaction or diffusion takes place, i.e., the subvolume and its neighbors.

We already saw in section 2 that care must be taken when transferring concepts from population-based to agent-based modeling. Nevertheless, agent-based models—which by definition feature independent entities—can be expected to contain a high level of locality.

One approach to exploit locality in agent-based simulation uses "spheres of influence" to determine which part of the model state is affected by a particular event (Theodoropoulos and Logan 1999). This information is then used to cluster highly interdependent agents on one computing node to reduce communications between distributed computing nodes (and thus the latencies that arise due to remote communications). A different example for the exploitation of locality in agent-based models is the Global-Scale Agent Model (GSAM) (Parker and Epstein 2011). The simulation scheme of this epidemiological model simulates only an *active* subset of the agent population, i.e., agents in certain infection states. New agents can only become active through events of active agents. As activation events occur predominantly in the immediate surroundings of an active agent, a large proportion of the agent population is unaffected by a distant disease outbreak. This population part can safely be disregarded by the simulation.

6 MODELING AND SIMULATION OF AGENT-BASED CTMC MODELS

From our previous considerations, we derive the following requirements to support modeling and simulation of agent-based CTMC models:

1. Separation of model definition and simulation (i.e., scheduling) algorithm
2. A model definition that abstractly describes state transitions and corresponding rate functions
3. An algorithm to determine the range of the effects of each event, to see which other events have to be rescheduled
4. Integration into an existing ABMS framework

We address these requirements with the software design that is described in the following. As a concrete ABMS framework to implement our design we chose Repast Simphony, although other software could be supportable similarly. Our proposed system provides a new way to define agents that are then executed by the underlying simulation system. All features of the ABMS framework, such as the GUI, visualizations, or analysis can also be used. The source code is available at git.informatik.uni-rostock.de/mosi/repast-sir-model-rulebased.

Our central idea is to introduce an additional simulation layer between the ABMS framework and the model definition. This layer provides interfaces for the definition of agents and rules that govern the agent's behavior. We opted for a rule-based language design to specify the agents' dynamics. Inspired by the modeling language ML3 (Modeling Language for Linked Lives) (Warnke et al. 2015), rules consist of three parts: a condition or guard function that determines which agents are exposed to the rule (taking the attributes of the agents into account), a function that calculates the waiting time of the rule (again in dependence of the involved agents and their attributes) and an effect function that is executed when the rule fires. The simulation layer keeps track of the agents in the model and their behavior rules, and schedules events accordingly. Whenever agents are affected by the execution of an event, the simulation layer reschedules their events. Thus, the definitions of the agent do not contain references to the scheduler (see Figure 4).

Our approach relies on some simple object-oriented concepts. First, an abstract class `Agent` is defined that must be implemented when creating a concrete agent class. Besides that, agent classes may be standard Java classes, and in particular contain attributes to store the state of a concrete agent. The simulation layer queries all agents for behavior rules via the interface defined by `Agent`. Similarly, `Rules` are described by an interface, which is implemented when defining concrete rules. Rule definitions can access attributes and methods of the agent class they are defined for. The predefined classes provide some auxiliary functions, which can be extended and reused across models. This makes the definition of agents and their behavior very succinct in comparison to the models with manual scheduling. Some features of Java 8, such as

```
1    public class SIRAgent extends Agent {
2
3    /* ... */
4
5    addRule(  () -> this.isInfectious(),
6              () -> exp(recoverRate),
7              () -> this.infectionState = InfectionState.RECOVERED);
8
9    addRule(  () -> this.isSusceptible(),
10             () -> exp(infectionRate * neighbours(SIRAgent.class).
11                       filter((SIRAgent agent) -> agent.isInfectious()).size()),
12             () -> this.infectionState = InfectionState.INFECTIOUS);
13   }
```

Figure 4: Model code snippet from a rule-based Repast implementation of an agent-based SIR model. The `addRule` method is provided by an abstract `Agent` class. It is called in the constructor of a concrete agent class. For the definition of condition, waiting time distribution, and effect the anonymous functions of Java 8 are exploited. The schedule is not accessed by the agent class.

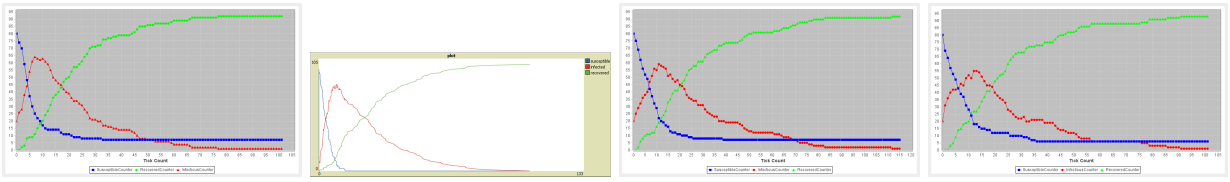

Figure 5: Output trajectories of the simple agent-based SIR model with an infection rate of 0.1 and a recovery rate of 0.05, produced with manual scheduling in Repast and Netlogo, and with the First Reaction Method and the Next Reaction Method in our simulation layer (from left to right).

lambda expressions, are applied to further minimize boilerplate code. However, as it is still possible to insert custom Java code, arbitrarily complex behavior can easily be implemented.

Within the simulation layer we implemented prototypical algorithms following the First Reaction Method as well as the Next Reaction Method. With both implementations we were able to reproduce the results of the model with manual scheduling (Figure 5). In general, the simulation algorithm does not schedule the actions of the agents directly. Instead, they are wrapped into a meta-event, which not only triggers the actual behavior of the agent, but also the rescheduling that might be necessary. Thus, the simulation algorithm is in control of all scheduling, whereas the agents are merely reacting to it (see Figure 6). Nevertheless, arbitrary Java code can be used, and agents are still able to directly access the schedule. However, this should be avoided, as possible side effects of events could be missed and prevent a correct rescheduling of events. Similarly, agents defined via the simulation layer can coexist with traditionally defined agents, but dependencies between them can not be managed by the simulation layer and, thus, may break the simulation algorithm.

For this simple model, our implementation of the Next Reaction Method reschedules events of all agents who are adjacent to the changed agent, regardless their state. Thus, some unnecessary rescheduling of unaffected recovery events occurs. We also assume that events of one agent affect each other. Consequently, the rules for each agent compete in a stochastic race and we only actually schedule the fastest event for each agent. The issue of determining more accurately which events affect each other, i.e., requirement 3 in the above list, will be addressed in future work.

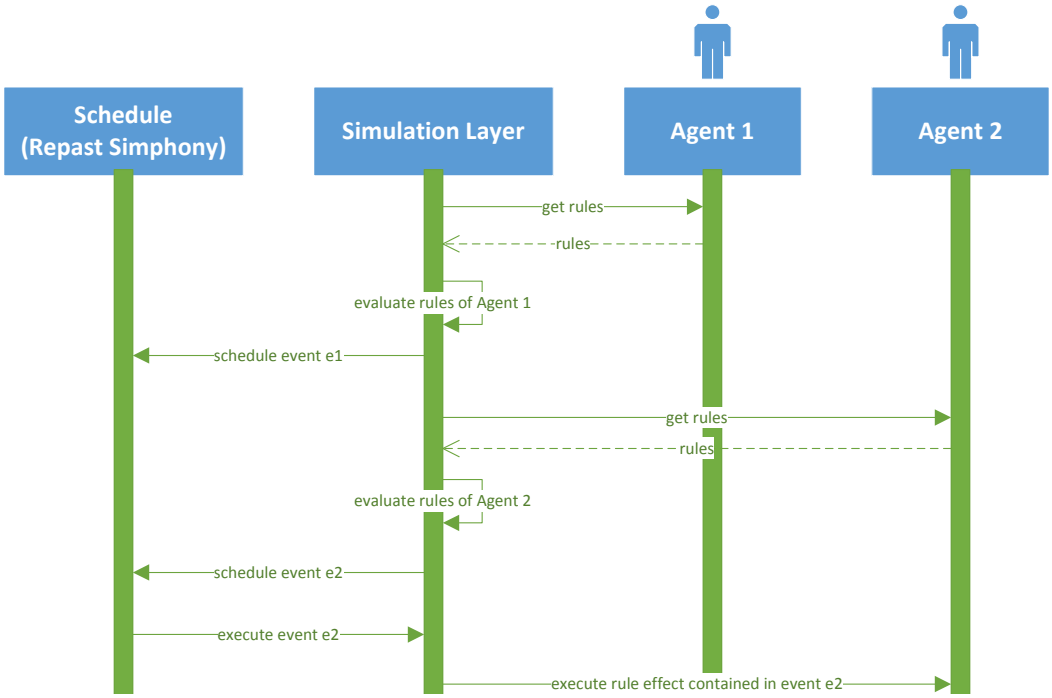

Figure 6: Sequence diagram illustrating the communication between the Repast Simphony core, the simulation layer and the agents in the Next Reaction Method. The agents and the schedule are not directly communicating.

7 CONCLUSION

We present an adaptation layer that enables a compact description of agent-based CTMC models in a style that resembles rule-based languages and allows the execution of agents in Repast Simphony. Two stochastic simulation algorithms, the Next Reaction Method and the First Reaction Method, have been realized. The Next Reaction Method relies on information about dependencies, i.e., which event has an impact on which other event. In the current implementation we exploit the concept of local interaction and assume that only direct neighbors in an interaction network possibly influence each other. For other influence patterns our approach would need to be adapted. Future work will be dedicated towards allowing to parameterize the algorithm for specific model classes or towards identifying influence patterns automatically. The First Reaction Method does not require any additional information, but will likely have a significantly inferior performance for larger models.

Using the presented approach, execution details that form a salient part of continuous-time agents in current ABMS frameworks are moved out of the model, which leads to more compact description of agents. However, agent definitions are still full Java classes and as such can contain arbitrarily complex code, e.g., the simulation algorithm can be disrupted by agents accessing the schedule directly. Thus, it is the responsibility of the modeler to adhere to the defined programming interface in using the layer. Future work might use stricter encapsulation to enforce this adherence more effectively.

Putting a layer on top of an existing ABMS framework like Repast Simphony rather than implementing the whole concept from scratch has several advantages. First, some functionality of the framework could directly be reused, in particular the schedule implementation. Second, Repast Simphony provides readily usable tools for observation, visualization, and analysis as well as bindings to other software. Third, the model could also be executed with another ABMS framework, if the presented approach has been adopted by the other framework as well. Our work brings together two different fields of modeling and simulation:

on the one hand CTMC-based modeling methods, which are typically compact and formally grounded, and on the other hand agent-based modeling and simulation, a widely adopted method in a variety of scientific application domains with well-established tool support.

ACKNOWLEDGMENTS

This research is partly supported by the German Research Foundation (DFG) via research grant UH-66/15-1.

REFERENCES

Allen, L. J. S. 2008. "An Introduction to Stochastic Epidemic Models". In *Mathematical Epidemiology*, Number 1945 in Lecture Notes in Mathematics, 81–130. Springer Berlin Heidelberg.

Allen, L. J. S., and G. E. Lahodny. 2012. "Extinction Thresholds in Deterministic and Stochastic Epidemic Models". *Journal of Biological Dynamics* 6:590–611.

Baier, C., B. R. Haverkort, H. Hermanns, and J.-P. Katoen. 2010. "Performance Evaluation and Model Checking Join Forces". *Communications of the ACM* 53 (9): 76–85.

Brown, R. 1988. "Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem". *Communications of the ACM* 31 (10): 1220–1227.

Carley, K. M., D. B. Fridsma, E. Casman, A. Yahja, N. Altman, L.-C. Chen, B. Kaminsky, and D. Nave. 2006. "BioWar: Scalable Agent-Based Model of Bioattacks". *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 36 (2): 252–265.

Ciocchetta, F., A. Degasperi, J. Hillston, and M. Calder. 2009. "Some Investigations Concerning the CTMC and the ODE Model Derived From Bio-PEPA". *Electronic Notes in Theoretical Computer Science* 229 (1): 145–163.

Cioffi-Revilla, C., and M. Rouleau. 2010. "MASON RebeLand: An Agent-Based Model of Politics, Environment, and Insurgency". *International Studies Review* 12 (1): 31–52.

Doob, J. L. 1945. "Markoff Chains–Denumerable Case". *Transactions of the American Mathematical Society* 58 (3): 455.

Elf, J., and M. Ehrenberg. 2004. "Spontaneous Separation of Bi-Stable Biochemical Systems Into Spatial Domains of Opposite Phases". *Systems Biology* 1:230–236.

Gibson, M. A., and J. Bruck. 2000. "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels". *The Journal of Physical Chemistry A* 104 (9): 1876–1889.

Gillespie, D. T. 1976. "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions". *Journal of Computational Physics* 22 (4).

Gillespie, D. T. 1977. "Exact Stochastic Simulation of Coupled Chemical Reactions". *The Journal of Physical Chemistry* 81 (25): 2340–2361.

Helms, T., R. Ewald, S. Rybacki, and A. M. Uhrmacher. 2015. "Automatic Runtime Adaptation for Component-Based Simulation Algorithms". *ACM Transactions on Modeling and Computer Simulation* 26 (1): 7:1–7:24.

Henzinger, T. A., B. Jobstmann, and V. Wolf. 2011. "Formalisms for Specifying Markovian Population Models". *International Journal of Foundations of Computer Science* 22 (04): 823–841.

Himmelspach, J., and A. Uhrmacher. 2007. "Plug'n Simulate". In *Proceedings of the 40th Annual Simulation Symposium*, 137–143.

John, M., C. Lhoussaine, J. Niehren, and C. Versari. 2011. "Biochemical Reaction Rules with Constraints". In *Programming Languages and Systems*, 338–357. Springer.

Klabunde, A., F. Willekens, S. Zinn, and M. Leuchter. 2015. *An Agent-Based Decision Model of Migration, Embedded in the Life Course - Model Description in ODD+D Format*. MPIDR Working Paper WP-2015-002. Max Planck Institute for Demographic Research.

Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. 2005. "MASON: A Multiagent Simulation Environment". *Simulation* 81 (7): 517–527.

North, M. J., N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko. 2013. "Complex Adaptive Systems Modeling with Repast Simphony". *Complex Adaptive Systems Modeling* 1 (1): 1–26.

Parker, J., and J. M. Epstein. 2011. "A Distributed Platform for Global-Scale Agent-Based Models of Disease Transmission". *ACM Transactions on Modeling and Computer Simulation* 22 (1): 2:1–2:25.

Railsback, S. F., S. L. Lytinen, and S. K. Jackson. 2006. "Agent-Based Simulation Platforms: Review and Development Recommendations". *Simulation* 82 (9): 609–623.

Sarjoughian, H. S., B. P. Zeigler, and S. B. Hall. 2001. "A Layered Modeling and Simulation Architecture for Agent-Based System Development". *Proceedings of the IEEE* 89 (2): 201–213.

Sheppard, C. J. R. and Railsback, S. 2015. "Time Extension for NetLogo (Version 1.2)". https://github.com/colinsheppard/time. Accessed July 2016.

Silverman, E., J. Bijak, J. Noble, V. Cao, and J. Hilton. 2014. "Semi-Artificial Models of Populations: Connecting Demography with Agent-Based Modelling". In *Advances in Computational Social Science*, Volume 11 of *Agent-Based Social Systems*, 177–189. Springer Japan.

Sneddon, M. W., J. R. Faeder, and T. Emonet. 2011. "Efficient Modeling, Simulation and Coarse-Graining of Biological Complexity With NFsim". *Nature Methods* 8 (2): 177–183.

Steiniger, A., and A. M. Uhrmacher. 2016. "Intensional Couplings in Variable-Structure Models: An Exploration Based on Multilevel-DEVS". *ACM Transactions on Modeling and Computer Simulation* 26 (2): 9:1–9:27.

Sweda, T. 2011. "Discrete Event Simulation Using Repast Java: A DiscreteEventSim Tutorial". https://code.google.com/archive/p/repast-demos/wikis/DiscreteEventSim.wiki. Accessed July 2016.

Theodoropoulos, G., and B. Logan. 1999. "A Framework for the Distributed Simulation of Agent-Based Systems". In *Proceedings of the 13th European Simulation Multiconference (ESM'99)*, 58–65: Citeseer.

Versari, C., and N. Busi. 2008. "Efficient Stochastic Simulation of Biological Systems with Multiple Variable Volumes". *Electronic Notes in Theoretical Computer Science* 194 (3): 165–180.

Wainer, G. A. 2009. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. 1st ed. Boca Raton, FL, USA: CRC Press, Inc.

Warnke, T., T. Helms, and A. M. Uhrmacher. 2015. "Syntax and Semantics of a Multi-Level Modeling Language". In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, 133–144: ACM.

Warnke, T., A. Steiniger, A. M. Uhrmacher, A. Klabunde, and F. Willekens. 2015. "ML3: A Language for Compact Modeling of Linked Lives in Computational Demography". In *Proceedings of the 2015 Winter Simulation Conference*, 2764–2775. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wilensky, Uri 1999. "NetLogo". http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Accessed July 2016.

Willekens, F. 2009. "Continuous-Time Microsimulation in Longitudinal Analysis". In *New Frontiers in Microsimulation Modelling*, edited by A. Zaidi, A. Harding, and P. Williamson, 413–436. Ashgate.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation*. 2nd ed. San Diego, CA, USA: Academic Press.

AUTHOR BIOGRAPHIES

TOM WARNKE is a Ph.D. student in the modeling and simulation group at the university of Rostock. His email address is tom.warnke@uni-rostock.de.

OLIVER REINHARDT is a Ph.D. student in the modeling and simulation group at the university of Rostock. His email address is oliver.reinhardt@uni-rostock.de.

ADELINDE M. UHRMACHER is professor at the Institute of Computer Science, University of Rostock and head of the modeling and simulation group. Her email address is adelinde.uhrmacher@uni-rostock.de.