Population-Based CTMCs and Agent-Based Models

Tom Warnke, Oliver Reinhardt, and Adelinde M. Uhrmacher
Institute of Computer Science, University of Rostock

Continuous-time agent-based modeling

- Social scientists develop continuous-time models
  - demographic events (marriage, childbirth, death)
  - decision processes (e.g., migration)
- Agent-based models are mostly implemented in ABMS frameworks (Repast Simphony, NetLogo, etc.)
- These frameworks lack support for continuous-time models
  - Solution 1: Develop an external domain specific language[1]
  - Solution 2: Integrate continuous-time modeling into ABMS frameworks

[1] T. Warnke, A. Steiniger, A. M. Uhrmacher, A. Klabunde, and F. Willekens. 2015. ML3: A Language for Compact Modeling of Linked Lives in Computational Demography. *WSC 2015*.

Continuous-time population-based modeling

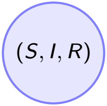
Example: An SIR model

- Three sub-populations of **S**usceptible, **I**nfectious, and **R**ecovered individuals
- Each model state is a triple (S, I, R)
- Two possible transitions:
  - A susceptible agents gets infected
  - An infectious agent recovers
- Exponentially distributed waiting time for each possible state transition

$\Rightarrow$ Continuous-Time Markov Chain (CTMC)

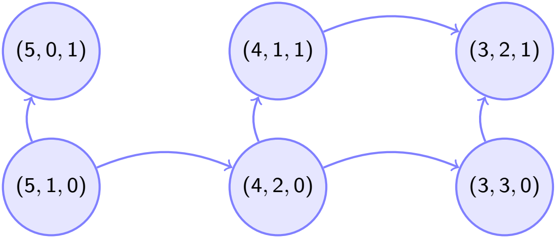
Formalisms for population CTMCs

State space and state transitions



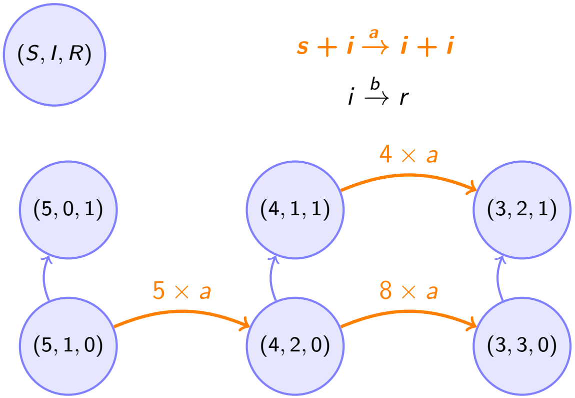
$$s + i \xrightarrow{a} i + i$$

$$i \xrightarrow{b} r$$



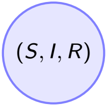
Formalisms for population CTMCs

State space and state transitions



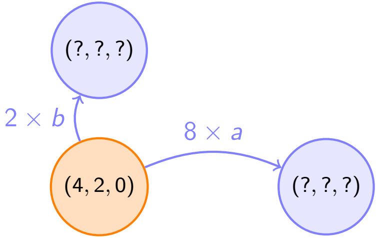
Formalisms for population CTMCs

Simulation and stochastic race



$$s + i \xrightarrow{a} i + i$$

$$i \xrightarrow{b} r$$



An agent-based continuous-time SIR model

- Agents are connected in a network
- Susceptible agents get infected after a stochastic waiting time based on the number of infected network neighbors
- Infected agents recover after a stochastic waiting time

SIR model in Repast Simphony

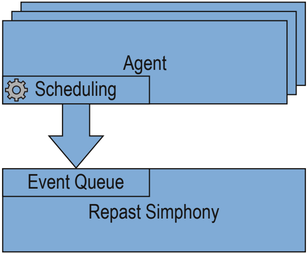
A small snippet of the behavior specification (about 50 lines)

```
private void scheduleInfection() {
  double currentTime = schedule.getTickCount();
  double infectiousNeighbors = getInfectiousNeighbors();
  if (infectiousNeighbors == 0) {
    scheduledEvent = null;
  } else {
    double rate = infectionRate * infectiousNeighbors;
    double waitingTime = RandomHelper.createExponential(rate).
         nextDouble();
    scheduledEvent = schedule.schedule(ScheduleParameters.createOneTime
         (currentTime + waitingTime), this, "getInfected");
  }
}
```

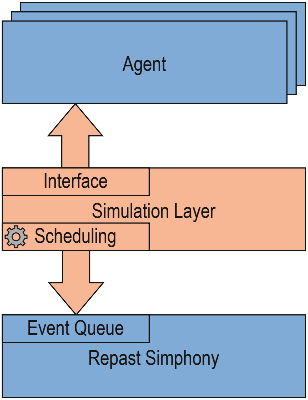
Assessment

- Repast Simphony provides a schedule object that allows inserting events in an event queue
- Continuous-time models require manually scheduling and retracting events
- The resulting model- and simulation-specific code is mixed
  - ⇒ Model is not readable
  - ⇒ Reusing code is hard

Scheduling in Vanilla Repast Simphony



Scheduling in Repast Simphony with the simulation layer



SIR model in Repast Simphony with the simulation layer

The complete behavior specification (10 lines)

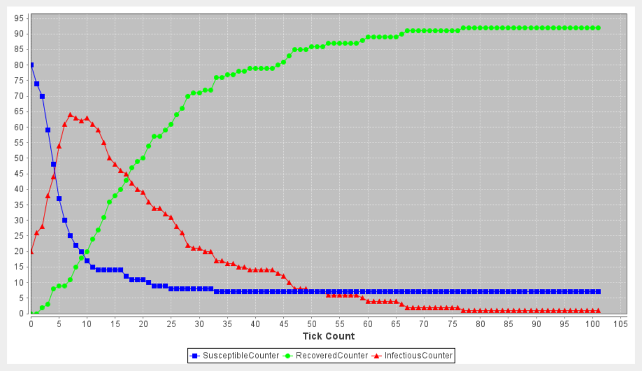
```
addRule(() -> this.isInfectious(),
        () -> exp(recoverRate),
        () -> this.infectionState = InfectionState.RECOVERED);

addRule(() -> this.isSusceptible(),
        () -> exp(infectionRate * neighbours(SIRAgent.class).
              filter((SIRAgent agent) -> agent.isInfectious()).
              size()),
        () -> this.infectionState = InfectionState.INFECTIOUS);
```

The simulation layer

- The simulation layer provides an interface with a domain-specific language (DSL) for succinct definition of agent behavior
- Agents can define their behavior as rules (guard, waiting time, effect)
- The simulation layer can query all agents for their behavior rules
  - to get all possible transitions from the current state
  - to construct (a part of) the CTMC
- Stochastic Simulation Algorithms in the simulation layer execute the CTMC
  - First Reaction Method (only schedule the globally first event)
  - Next Reaction Method (schedule several events and reschedule if necessary)

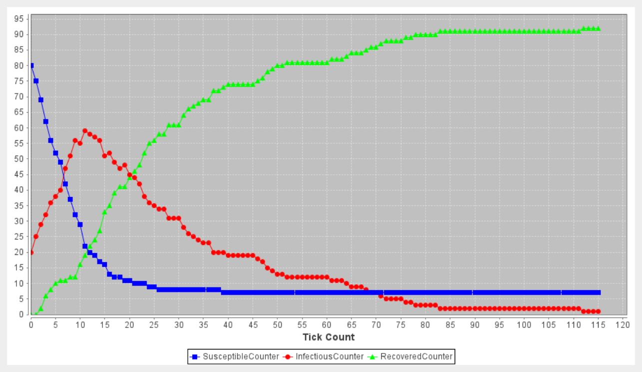
Output

Manual scheduling



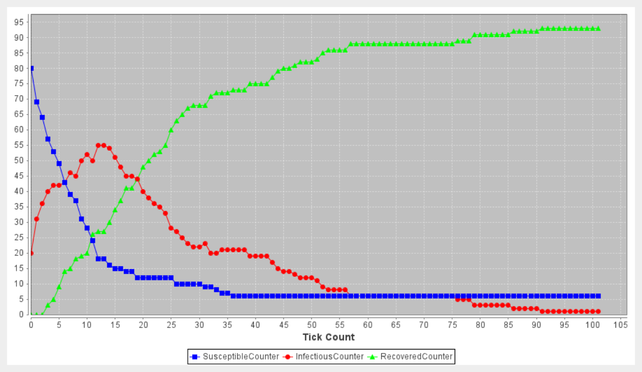
Output

First Reaction Method



Output

Next Reaction Method



An embedded DSL for modeling

Reflections and lessons learned

- Separate problem definition (model) from execution code (simulators)
  - ⇒ Multiple simulation algorithms are applicable and can be reused
- No reference to the schedule in the model
  - ⇒ Succinct, easily editable and reusable model
- Rule-based syntax (conditions, waiting time, effect) and CTMC semantics
  - ⇒ Semantically sound simulation with SSA-style execution algorithms
- Simulation efficiency depends on exploiting locality
  - ⇒ More work on model analysis needed