A DSL for Continuous-Time Agent-Based Modeling and Simulation

Tom Warnke
Institute of Computer Science, University of Rostock

Continuous-time agent-based modeling

Limitations in the state of the art

- Agent-based models are mostly developed in ABMS frameworks (Repast Simphony, Netlogo, etc.)
- These frameworks support time-stepped models very well
- However, many problems can be modeled better in continuous time
- Continuous-time models in ABMS frameworks require manual scheduling
- The resulting model- and simulation-specific code is mixed
  - $\Rightarrow$ Model is not readable
  - $\Rightarrow$ Reusing code is hard

An agent-based continuous-time SIR model

An example

- Agents are either susceptible, infected or recovered
- Agents are connected in a network
- Initially, some agents are infected
- 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

Before

A small snippet of the behavior specification

```
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");
  }
}
```

After

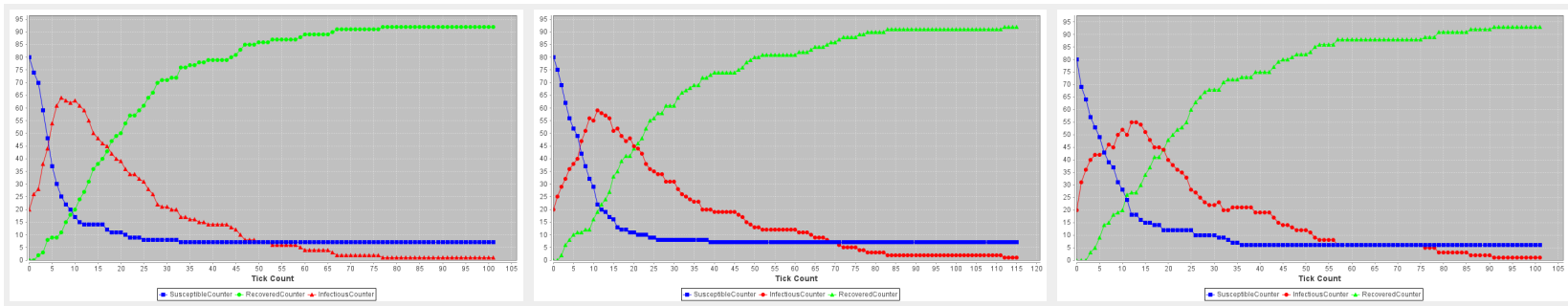
The complete behavior specification

```
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);
```

Output

Manual scheduling, First Reaction Method, 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 and readable model
- Rule-based syntax (conditions, waiting time, effect) and CTMC semantics
  - ⇒ Semantically sound simulation with SSA-style execution algorithms
- Efficiency depends on exploiting locality
  - ⇒ More work on model analysis needed