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.

A DSL FOR CONTINUOUS-TIME AGENT-BASED MODELING AND SIMULATION

Tom Warnke

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

ABSTRACT

Most state-of-the-art agent-based modeling and simulation (ABMS) frameworks offer a way to describe agent behavior in a programming language. Whereas these frameworks support easy development of time-stepped models, continuous-time models can only be implemented by manually scheduling and retracting events as part of the agent behavior. To facilitate a separation of concerns into model- and simulation-specific code for continuous-time ABMS, we propose an embedded domain-specific language, which allows describing agent behavior concisely, and corresponding simulation algorithms, which allow executing continuous-time models. The language style and the algorithms are adapted from rule-based modeling languages for Continuous-Time Markov Chains and Stochastic Simulation Algorithm variants. We implemented prototypes of the modeling language and simulation algorithms based on Repast Simphony.

1 A DSL FOR CONTINUOUS-TIME AGENT-BASED MODELING AND SIMULATION

Domain-specific languages (DSLs) have successfully been applied to a plethora of problems, for example as modeling languages. Their main property is the focus on a specific purpose, leading to syntax and semantics that are tailored to the task at hand. DSLs can be designed as external or embedded languages, with different advantages and disadvantages (Van Deursen, Klint, and Visser 2000).

External DSLs with custom grammars and algorithms provide total freedom to the language designer. However, custom implementations of external DSLs are also cumbersome and error-prone. Depending on the expressive power of the language, execution complexity can pose a challenge as well. Embedded DSLs, on the other hand, are integrated in a host language and build upon its syntax and semantics. Thus, the code can be compiled with the host language compiler, which typically involves heavy optimization and leads to more efficient execution. For many programming languages techniques to circumvent the constraints on the syntax of embedded DSLs have been developed. In particular, the paradigm of functional programming (as implemented in languages such as Scala, Groovy, or recent Java versions) has proved to facilitate the development of DSLs. In this work, we explore possibilities to exploit an embedded DSL as a modeling language for continuous-time agent-based modeling and simulation (ABMS).

Many well-established frameworks for ABMS, such as Repast Simphony and NetLogo, provide interfaces for the user to "program" the model by defining the behavior of agents in a programming language. They offer DSL-like syntactical shortcuts for models that shall be executed with fixed time steps, making code for such models very concise. However, to model stochastic processes realistically, many applications require continuous-time modeling. The ABMS frameworks currently support continuous-time modeling only by providing access to an event schedule. The modeler is required to manually schedule events and retract events that have been invalidated. Consequently, the actual model code (i.e., code defining the agent behavior) is diluted by execution code (i.e., code to manage events). Mixing code for model and simulation in this way is error-prone and hampers reuse of code.

```
1    addRule(  () -> this.isInfectious(),
2              () -> exp(recoverRate),
3              () -> this.infectionState = InfectionState.RECOVERED);
4    }
```

Figure 1: This model code snippet defines a behavior rule for an agent. A rule consists of a condition, a waiting time expression, and a rule effect. The rule shown here models the recovery of an infected agent.

A separation of concerns, on the other hand, has been shown to be beneficial in modeling and simulation and also generally in software design. Rule-based modeling languages, for example in systems biology, are a prime example of this concept. They are typically implemented as external DSLs, accompanied by simulation algorithms to execute a defined model. The basis for many rule-based modeling languages and their simulation algorithms are Continuous-Time Markov Chains (CTMCs) and the stochastic simulation algorithm (SSA) (Warnke, Helms, and Uhrmacher 2015). A plethora of SSA-based algorithm has been proposed, and CTMCs have been successfully applied as models for population-based processes. The fundamental idea here is that all state transitions (i.e., changes of the population sizes) compete in a "stochastic race". However, when transferring this idea to agent-based models, the resulting CTMC is much larger than typical population-based CTMCs. The size renders most of the basic SSA variant useless, and SSA variants that exploit locality are required. Instead of constantly reevaluating all rules to obtain state transitions, these variants maintain unchanged state transitions, thus saving computation time.

To facilitate continuous-time agent-base modeling and simulation, we developed a *simulation layer* for Repast Simphony (North et al. 2013). It allows defining agent behavior concisely in a rule-like embedded DSL by exploiting Java 8's lambda expressions (figure 1). The rules can contain arbitrarily complex functions that will be compiled and, thus, executed efficiently. The agent behavior rules are used by simulation algorithms that are implemented in the simulation layer. Thus, the model description is completely separated from the schedule-specific code. We also implemented a prototypical simulation algorithm that exploits locality. The main challenge here is to assess the locality of agent actions based on the agents' behavior definitions. Currently, we are assuming that agents are connected in a social network and are only influenced by actions of their direct network neighbors. Thus, after an action of an agent, only its direct neighbors need to be updated (i.e., their rules need to be reevaluated), leading to an efficient execution of the model.

We applied our simulation layer to a simple continuous-time agent-based SIR model and were able to reproduce the results from similar models with manual event scheduling and retraction. Thus, our proposed approach leads to expressive and efficient, but concise and readable model descriptions. It may serve as an example for the usefulness of embedded DSLs as modeling languages. A more thorough discussion of the underlying ideas can be found in the full paper (Warnke, Reinhardt, and Uhrmacher 2016).

ACKNOWLEDGEMENTS

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

REFERENCES

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.

Van Deursen, A., P. Klint, and J. Visser. 2000. "Domain-Specific Languages: An Annotated Bibliography". *Sigplan Notices* 35 (6): 26–36.

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

Warnke, T., O. Reinhardt, and A. M. Uhrmacher. Appearing 2016. "Population-Based CTMCs and Agent-Based Models". In *Proceedings of the 2016 Winter Simulation Conference*.