A Language for Agent-Based Discrete-Event Modeling and Simulation of Linked Lives

OLIVER REINHARDT, TOM WARNKE, and ADELINDE M. UHRMACHER, University of Rostock, Institute for Visual and Analytic Computing, Germany

In agent-based modeling and simulation, discrete-time methods prevail. While there is a need to cover the agents' dynamics in continuous time, commonly used agent-based modeling frameworks offer little support for discrete-event simulation. Here we present a formal syntax and semantics of the language ML3 (Modeling Language for Linked Lives) for modeling and simulating multi-agent systems as discrete-event systems. The language focuses on applications in demography, such as migration processes, and considers this discipline's specific requirements. These include the importance of life courses being linked, and the age-dependency of activities and events. The developed abstract syntax of the language combines the metaphor of agents with guarded commands. Its semantics is defined in terms of Generalized Semi-Markov Processes. The concrete language has been realized as an external domain-specific language. We discuss implications for efficient simulation algorithms and elucidate benefits of formally defining domain-specific languages for modeling and simulation.

ACM Reference Format:
Oliver Reinhardt, Tom Warnke, and Adelinde M. Uhrmacher. 2021. A Language for Agent-Based Discrete-Event Modeling and Simulation of Linked Lives. *ACM Trans. Model. Comput. Simul.* 1, 1 (June 2021), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Agent-based modeling and simulation has become an established method to understand social processes in heterogeneous populations [20], more recently also in demography [6]. Demographic phenomena such as migration can be modeled as systems of agents that live "linked lives", interacting with each other via social ties [40, 41]. To support modeling such systems, we have presented the modeling language for linked lives (ML3) [64, 66]. However, so far ML3 has only been presented informally. The syntax is illustrated with a set of example models, and the semantics is given as a reference simulator implementation [56]. In this paper, we present for the first time a formal definition of ML3 and, thus, provide an abstract and unambiguous mapping of ML3 models to the stochastic processes they describe. We relate the simulation algorithms to these stochastic processes, bridging the gap between the reference implementation and the formal semantics.

The design of ML3 is characterized by specific requirements from the application domain of agent-based modeling in demography [66]. In particular, ML3 is tailored to models with (1) discrete events in continuous time, (2) age-specific and deterministic events, and (3) agents that are connected via network edges. These design decisions distinguish ML3 from existing approaches. Moreover, as we show in this paper, the design of ML3 is also reflected in the formal language definition, which therefore differs from the usual style of defining simulation modeling languages formally (also see

Authors' address: Oliver Reinhardt, oliver.reinhardt@uni-rostock.de; Tom Warnke, tom.warnke@uni-rostock.de; Adelinde M. Uhrmacher, adelinde.uhrmacher@uni-rostock.de, University of Rostock, Institute for Visual and Analytic Computing, Albert-Einstein-Straße 22, Rostock, MV, 18059, Germany.

Manuscript submitted to ACM

section 6). In particular, the flat network structure of model states precludes a hierarchical decomposition of models, as used, for example, in DEVS-based approaches [69]. Deterministically timed and periodic events as supported by ML3 are not expressible with continuous-time Markov chains (CTMCs), which are a popular semantic domain for discrete-event modeling languages [8].

Our main contribution in this paper is a formal and precise definition of the syntax and semantics of ML3. The remainder of the paper is organized as follows. First, we illustrate the design of ML3 by showing how a previously published demographic model benefitted from being expressed in ML3. Section 3 then contains the core of this paper: the formal language definition. Starting with a simple CTMC-based language modeling networks of agents, we illustrate how introducing demography-specific features leads to more complex process models. We arrive at a mapping of ML3 models to Generalized Semi-Markov Processes, allowing age-dependent transition rates and deterministic events. In the sections 4 and 5 we then describe how the formal definition relates to the software implementation of ML3. In particular, we show how the differently complex stochastic processes are reflected in different simulation algorithms. The paper concludes with a review of related modeling formalisms and a discussion of ML3 and the challenges of integrating discrete-event simulation and agent-based modeling with a focus on applications in demography.

2 MOTIVATING EXAMPLE

To motivate the need for a language and its requirements, and to give an intuition about the type of systems we are interested in modeling, we will use several excerpts of a simulation model developed by Klabunde et al. [40, 41], a discrete-event model of potential migrant's decision making with event scheduling in continuous time. The authors originally implemented their model in NetLogo using the aforementioned extension for continuous-time discrete-event simulation. An earlier implementation of that model in ML3, described in more detail, can be found in [64].

Klabunde et al. model how potential migrants make the decision to migrate, in the case of irregular migration from Senegal to France. At the core of the model is a multi-stage decision process. Therein, the agent will form an intention to migrate based on Ajzen's Theory of Planned Behavior [1], taking a variety of economic, social, and environmental factors into account. This intention governs the migrants path through a multi-stage decision process, from having formed an intention, through planning, preparing and finally migrating. In each of these stages the migrant might succeed and advance to the next stage, fail and be set back to the beginning of the process, or leave the process entirely, abandoning the decision.

Figure 1 shows a rule from this model that governs the final step of this process, the actual migration attempt, implemented in ML3. In ML3 rule define the behavior of agents, and consist of four parts (formally defined in Section 3). The first part (line 1), defines which type of agents (here: persons) the rule applies to. The second part, the guard (line 2), specifies preconditions for this behavior. To migrate, the person must be in the migration decision process, have a positive intention, must be in the preparation stage of the process, and have sufficient funds. With a certain rate (the third part, defined in line 3), they will then attempt to migrate. The final part of the rule (lines 4-8) defines the effect of that behavior, i.e., their attempt to migrate. Depending (stochastically) on the intensity of border enforcement this attempt might succeed or fail. In case of success, they will migrate, which is a complex endeavor in its own right. In case of failure, they are reset to the beginning of the process. This type of complex decision making is typical for certain agent-based models in the social sciences, that aim to explain social phenomena as the product of individual decision making, e.g., [32], [38], [5]. The language must therefore allow for the modeling of complex behavior in a social context.

While not being universal, it is common for demographic models to track individuals through their whole life course, e.g., [51], [5], [4]. By progressing through their life courses, they experience a variety of demographic events. As an

```
1  Person
2  |  ego.inMigrationProcess(), ego.migrationIntention() >= 0, ego.migrationStage
       = "preparation", ego.canAffordMigration()
3  @  ego.migrationAdvancementRate()
4  -> if (random() <= ego.successProbability()) then
5       ego.migrate()
6     else
7
8
9       ego.failMigration()
10    end
```

Fig. 1. Stochastic rule governing the final step of the decision process in [41], slightly adapted from [64].

example, Figure 2 shows the fertility process from the migration decision model. Women in a certain age range (line 2) may give birth to a child with a certain rate (line 3), depending on cohort, age, and number of children. A new child agent is then created and initialized (lines 4-10). A language for demographic models must allow for the specification of these demographic processes easily. In particular, as these processes are often age-dependent, e.g., fertility rates vary with the parents' ages, it must be possible to specify age-dependent behavior.

```
1  Person
2  |  ego.sex = "f", ego.age >= minFertilityAge, ego.age < (maxFertilityAge + 1)
3  @  ego.birthRate()
4  -> ?child := new Person(
5       sex := ["m", "f"].random(),
6       parents := [ego] + if ego.isMarried() then [ego.partner] else [],
7       migrationStartAge := normal(meanMigrationStartAge, 1)
8     )
9     ?child.moveToHousehold(ego.household)
10    ?child.moveToAddress(ego.address)
```

Fig. 2. Stochastic rule governing fertility in [41], slightly adapted from [64].

The processes shown above are all modeled with stochastic timing. While this is appropriate for the decision process and fertility, not every process in a social system can be reasonably modeled as such. For example, in the migration model, incomes are paid and received monthly. Also, people become legally adult at a certain specified age, which allows them to behave differently (see Figure 3). In addition to stochastic behavior, a language for such models must allow for the modeling of such non-stochastic events.

3 ABSTRACT SYNTAX AND SEMANTICS

We formally define the syntax and semantics of ML3 in three steps, introducing three "levels" of the language, each expanding on the one before. The first allows one to define the agents behavior using stochastic rules with exponential rates, leading to a classical homogeneous Continuous-time Markov Chain (CTMC) semantics. At this level, the language already allows for modeling complex decision processes in continuous time, the first of the requirements described

```
Person
|  ego.status = "child"
@  age ageOfAdulthood
-> ego.drawIncome()
   ego.status := "adult"
```

Fig. 3. Non-stochastic rule governing becoming an adult in [41], slightly adapted from [64].

above. At the second level, we additionally allow that the exponential rate may depend on time and age, which results in the CTMC being non-homogeneous. At this level, the language will allow for the modeling of the age-dependent processes common in life-course models. Finally, the third level will allow non-stochastic rules with fixed or periodic execution times. As we go beyond of exponential distributions, the resulting process is no longer a CTMC. Hence, we define the semantics of the third level in terms of a Generalized Semi-Markov Process.

For each level, we begin with a definition of the abstract syntax, followed the semantics. We begin with some preliminary definitions of central concepts and notations.

3.1 Preliminaries

In the following, we assume there is a given set of basic types $\mathcal{B} \supseteq \{\textbf{real}, \textbf{int}, \textbf{bool}\}$. For each type $\tau \in \mathcal{B}$ there exists a corresponding set of values $\mathcal{V}(\tau)$, with $\mathcal{V}(\textbf{real}) = \mathbb{R}$, $\mathcal{V}(\textbf{int}) = \mathbb{Z}$ and $\mathcal{V}(\textbf{bool}) = \{T, F\}$.

3.1.1 Agents. The fundamental entity of any agent-based model are **agents**. In ML3, not only individual persons, but also higher level actors such as households or the government must be modeled as agents if they have a state or behavior. All agents are of a type $\alpha \in \mathcal{A}$, where $\mathcal{A}$ denotes the set of all agent types. The agent's type determines its (typed) attributes, denoted as $\alpha = (att_1 : \tau_1, \ldots, att_n : \tau_n)$. The attribute names att_i are mutually distinct elements of a set of attribute names, and the τ_i are from the set of basic types $\mathcal{B}$. Among the attributes, each agent type α has the following: First, `birth` : **`real`**, the time of the agent's birth, which together with the current time defines the agent's age. This allows for specifying age-dependent behavior. Second, `alive` : **`bool`**. We will distinguish agents that are alive - and may exhibit independent behavior - and agents that are dead - who may not act on their own. However, other agents' behavior might can be influenced by dead agents (they can still access their attributes), and a dead agent's position in the social network might still matter (e.g., an agent's siblings are the agents that share the same parents - still being connected to the parents is therefore necessary to identify siblings). And third, `id` : **`int`**, an identifier, which is assumed to be unique over all agents, allowing for the distinction between agents that are identical in all other attributes.

We define agents a of an agent type α to be mappings of the attribute names defined by the type to corresponding values. We use $a.att$ for the value of a's attribute att, and $a.att \in \mathcal{V}(\tau)$ if a's type α contains $att : \tau$. For each agent-type $\alpha \in \mathcal{A}$ we define the set of values (the set of possible agents of this type) $\mathcal{V}(\alpha)$ as the set of all such mappings.

Agent types and agents are more closely comparable to classes and objects in object-oriented programming languages than to records and record values as in functional programming languages (cf. [54, chapter 18]). In particular, agents have an identity through the implicit `id` attribute. Contrary to object-oriented programming, however, we do not consider encapsulation of an agent's internal state here—all attributes are publicly accessible for other agents. Access rules for agent attributes could be defined and enforced by defining appropriate expression and statement languages (see section 3.1.5), for example with object-oriented getters and setters.

3.1.2 Links. As we need to model behavior that is dependent on an individual's social environment, we need to model this environment [29]. In ML3, this is realized with *links*. Links are bidirectional relations between agents of certain types, e.g., between parents and their children, people and their friends, or a household and its members. Potential links are defined by a set of link definitions $\mathcal{K}$. A specific link definition is denoted as $\kappa = (lnk_1 : (\alpha_1, card_1), lnk_2 : (\alpha_2, card_2)) \in \mathcal{K}$, with the lnk_i being mutually distinct elements of a set of role names, the $\alpha_i \in \mathcal{A}$, and the $card_i \in \{1, \mathsf{n}\}$. The link definition κ describes that each agent of type α_1 is linked to $card_2$-many agents of type α_2, that are referred to by the role name lnk_2. In the opposite direction, each agent of type α_2 is linked to $card_2$-many agents of type α_1, that are referred to by the role name lnk_1. Thereby $card_i = 1$ denotes up to one link partner, while $card_i = \mathsf{n}$ denotes that any (non-negative) number of link partners is possible. For example, a link between parents and their children would be represented as $\kappa = (children : (Person, \mathsf{n}), parents : (Person, \mathsf{n}))$. Figure 4 gives an example of a realization of this link with two parents and three children.

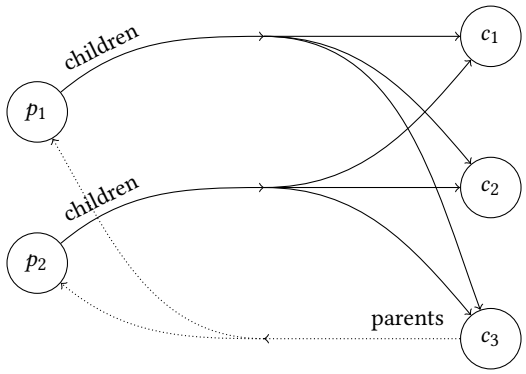

Fig. 4. Example of state of a model with a parents-children link. The agents p_1 and p_2 are parents; c_1, c_2 and c_3 are their children. Hence, the outgoing *children* link of p_1 and p_2 points to c_1, c_2 and c_3. The link direction from c_1 and c_2 to their parents is omitted, but analogous to the one of c_3.

3.1.3 Types. Given the basic types $\mathcal{B}$ and the agent types $\mathcal{A}$ we define a set of (all) types $\mathcal{T}$ as follows via induction:

- $\tau \in \mathcal{B} \Rightarrow \tau \in \mathcal{T}$
- $\alpha \in \mathcal{A} \Rightarrow \alpha \in \mathcal{T}$
- $\tau \in \mathcal{T} \Rightarrow \{\tau\} \in \mathcal{T}$

For the third case, we define the set of values as the power set of $\mathcal{V}(\tau)$: $\mathcal{V}(\{\tau\}) = \mathcal{P}(\mathcal{V}(\tau))$. With this definition, there is a $\mathcal{V}(\tau)$ defined for every $\tau \in \mathcal{T}$.

3.1.4 States. Model states are directed hypergraphs, more specifically forward-hypergraphs [24], i.e., ones in which the origin of all edges is only a single node, with agents represented by nodes and links between agents by edges. Edges are labeled with the link's role name. We will use $\sigma = (A, L, l)$ to denote states with the set of agents (nodes) A, the set of links (hyper-edges) L and the edge-labeling l. We will use $\mathcal{S}$ for the state-space of the model.

Given a state $\sigma = (A, L, l)$ we define $\alpha(\sigma) \subseteq A$ to be the set of agents in state σ that have type α.

3.1.5 Expressions and Statements. We will define our language abstractly, making use of expression languages and statement languages at appropriate places.

We have (typed) expressions languages $\mathbb{E}$ and $\mathbb{E}_t$, with $\mathbb{E} \subset \mathbb{E}_t$. Any $e : \tau \in \mathbb{E}$ is an expression that evaluates to values in $\mathcal{V}(\tau)$, given a model state and an agent that serves as an evaluation context (i.e., the value of ego in the above examples). The expression language may, for example, allow for the usual arithmetic and logical expressions, access to the agent's attributes, and to its links. For each typed expression $e : \tau \in \mathbb{E}$ there is an evaluation function $[\![e]\!]_{\mathbb{E}} : \mathcal{S} \times \mathcal{A} \to \mathcal{V}(\tau)$ that performs this evaluation. Expressions in $\mathbb{E}_t$ may also depend on the current simulation time, and for each $e : \tau \in \mathbb{E}_t$ there is an evaluation function $[\![e]\!]_{\mathbb{E}_t} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}_+ \to \mathcal{V}(\tau)$ with an additional time parameter. We require that time-independent expressions (in $\mathbb{E}$) retain their value when interpreted as a time-dependent expression, i.e., for all $e : \tau \in \mathbb{E}$, $\sigma \in \mathcal{S}$, $a \in \mathcal{A}$, $t \in \mathbb{R}_+$:

$$[\![e]\!]_{\mathbb{E}}(\sigma, a) = [\![e]\!]_{\mathbb{E}_t}(\sigma, a, t) \tag{1}$$

Similarly, given are statement languages $\mathbb{S}$ and $\mathbb{S}_t$, with $\mathbb{S} \subset \mathbb{S}_t$. For each $s \in \mathbb{S}$ we have an evaluation function $[\![s]\!]_{\mathbb{S}} : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to [0, 1])$, that, given the current state and an agent, returns a probability mass function for the distribution of successor states. This reflects the fact that the result of events might be stochastic, as in the examples in Figure 1, where the migration attempt might fail stochastically, or Figure 2, where the sex of the child is chosen at random. For each $s \in \mathbb{S}_t$ we have an evaluation function $[\![s]\!]_{\mathbb{S}_t} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}_+ \to (\mathcal{S} \to [0, 1])$ of the same kind, but with an additional time parameter. As we did for expressions, we require that time-independent statements retain the value when interpreted as time-dependent statements, i.e., for all $s : \tau \in \mathbb{S}$, $\sigma \in \mathcal{S}$, $a \in \mathcal{A}$, $t \in \mathbb{R}_+$:

$$[\![s]\!]_{\mathbb{S}}(\sigma, a) = [\![s]\!]_{\mathbb{S}_t}(\sigma, a, t) \tag{2}$$

Alternatively, the distinction between time-dependent and time-independent expressions could be realized via the type system. Then time-independent expressions are a subtype of time-dependent expressions [54, chapter 15]. Combining a time-dependent and a time-independent expression always yields a time-dependent expression. However, here we decided to keep the exact expression and statement languages, and as a consequence their type system, unspecified, and therefore exchangeable. When defining the concrete syntax of expressions or statements, syntactic conventions of existing object-oriented programming languages can be adopted (see section 5.2).

3.2 Core: Homogeneous CTMC

Fundamentally, all agents' behavior shall be described by stochastic rules, as shown in the example in Figure 1. In Figure 5 this structure is formally defined. Each rule consists of four parts: First, α, the type of agents this rules applies to (Figure 1 line 1). Second, e_1, a guard expression that describes which of those agents the rule applies to (line 2). Third, e_2, a rate expression that describes when and how often the rule will be applied (line 3). And finally, s, the effect that is triggered when the rule is executed, given as a statement (lines 4-8). A model is then nothing but a sequence of such rules.

We define the semantics of the modeling formalism on the basis of Continuous-time Markov Chains (CTMCs). Thereby, we have to map models to the CTMCs they describe. We will follow the state-to-function transition system approach introduced by De Nicola et al. [18].

Definition 3.1 (Continuous-time Markov Chain, after [13]). Let $\mathcal{S}$ be a countable state space. The $\mathcal{S}$-valued stochastic process $\{X(t)\}_{t \geq 0}$ is called a Continuous-time Markov Chain (CTMC), if for all $\sigma, \sigma', \sigma_1, \ldots, \sigma_k \in \mathcal{S}$, all $\Delta t, t, t_1, \ldots, t_k > 0$ with $t_l \leq t$ for all $l \in [1, k]$:

$$P(X(\Delta t + t) = \sigma' \mid X(t) = \sigma, X(t_i) = \sigma_1, \ldots, X(t_k) = \sigma_k) = P(X(\Delta t + t) = \sigma' \mid X(t) = \sigma) \tag{3}$$

$$
\begin{array}{lcll}
m & ::= & r\ m & \text{(Rule)} \\
 & | & \epsilon & \text{(Empty)} \\
 & & & \\
r & ::= & \alpha : e_1 \xrightarrow{e_2} s & \text{(Exp)} \\
 & \textbf{where} & \alpha \in \mathcal{A},\ e_1 : \texttt{bool} \in \mathbb{E},\ e_2 : \texttt{real} \in \mathbb{E},\ s \in \mathbb{S} &
\end{array}
$$

Fig. 5. Abstract syntax for core-level ML3 models (m) and rules (r). A model is a sequence of rules. Each rule consists of an agent type, a guard, a rate, and an effect.

If the right side of the equation is independent of t, i.e., for all $t \in \mathbb{R}$

$$P(X(\Delta t + t) = \sigma' \mid X(t) = \sigma) = P(X(\Delta t) = \sigma' \mid X(0) = \sigma) \tag{4}$$

the CTMC is called *homogeneous*, otherwise it is called *non-homogeneous*.

We characterize a homogeneous CTMC with their *transition rate matrix* $Q : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$. A transition from σ to σ' is possible, if $Q(\sigma, \sigma') > 0$. Note that we differ slightly from the definition of a CTMCs infinitesimal generator in that we allow $Q(\sigma, \sigma) > 0$, i.e., transitions to the same state may occur. This makes the following definitions a lot easier, without changing the character of the stochastic process [17].

To define Q, we define an operator $\rightarrowtail$ that maps tuples consisting of a model and a state to functions $S \to \mathbb{R}_+$. Let m be a model and σ the current state, with $\langle m, \sigma \rangle \rightarrowtail \mathscr{F}$. Then $\mathscr{F}$ is the row of the model's transition rate matrix that corresponds to the current state σ, and for all states σ': $Q(\sigma, \sigma') = \mathscr{F}(\sigma')$.

We define the behavior of a whole model by defining the behavior of single rules, using an operator $\rightarrowtail_r$ that serves the same purpose as $\rightarrowtail$, but operates on single rules instead of complete models. The semantics of the model is the combined behavior of all rules. The behavior of a single rule will in turn be defined by first defining the behavior of this rule applied to a single agent, using an operator $\rightarrowtail_i$ (i for instance), and combining the results for all applicable agents.

To enable a succinct definition of $\rightarrowtail$, we introduce to following notation for functions $\mathcal{S} \to \mathbb{R}_+$:

- We define [] to be the function $\mathcal{S} \to \mathbb{R}_+$ that is constant 0.
- We define $f \oplus g$ with $f, g : \mathcal{S} \to \mathbb{R}_+$ as the function that is gained by pointwise addition of f and g, i.e., $(f \oplus g)(\sigma) = f(\sigma) + g(\sigma)$. Whenever we use $\sum$ with functions $\mathcal{S} \to \mathbb{R}_+$, it refers to this pointwise summation.
- We define $c \odot f$ with $c \in \mathbb{R}_+$ and $f : \mathcal{S} \to \mathbb{R}_+$ as the function that is obtained by scalar multiplication of f with c, i.e., $(c \odot f)(\sigma) = c \cdot f(\sigma)$.

The definition of $\rightarrowtail$, $\rightarrowtail_r$ and $\rightarrowtail_i$ is shown in Figure 6. An empty model (see the rule labeled empty **Empty**) with no rules yields no transitions at all. Hence, for an empty model, all successor states are reached with transition rate 0, and the transition rate function yielded by $\rightarrowtail$ is constant 0. In a model with at least one rule (**Rule**), every rule yields some transition rate functions. These are added pointwise (i.e., we add the rate separately for each possible successor state) to yield the transition rates for the whole model. If the first rule yields the transition rate function $\mathscr{R}$, and the rest of the model yields $\mathscr{M}$, the complete model yields $\mathscr{R} \oplus \mathscr{M}$. If multiple rules yield the same successor states, this pointwise addition adds the transition rates. If they yield different states, the pointwise addition keeps them separate. Figure 7 illustrates this.

$$\textbf{(Empty)}\ \frac{}{\langle \epsilon, \sigma \rangle \rightarrowtail []} \qquad \textbf{(Rule)}\ \frac{\langle r, \sigma \rangle \rightarrowtail_r \mathscr{R} \qquad \langle m, \sigma \rangle \rightarrowtail \mathscr{M}}{\langle r\ m, \sigma \rangle \rightarrowtail \mathscr{R} \oplus \mathscr{M}}$$

$$\textbf{(Exp)}\ \frac{\sum_{\{(a,\mathscr{I})\ |\ a \in \alpha(\sigma), \langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a \rangle \rightarrowtail_i \mathscr{I}\}} \mathscr{I} = \mathscr{R}}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma \rangle \rightarrowtail_r \mathscr{R}}$$

$$\textbf{(Instance-D)}\ \frac{a.\texttt{alive} = F}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a \rangle \rightarrowtail_i []} \qquad \textbf{(Instance-F)}\ \frac{a.\texttt{alive} = T \qquad [\![e_1]\!]_{\mathbb{E}}(\sigma, a) = F}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a \rangle \rightarrowtail_i []}$$

$$\textbf{(Instance-T)}\ \frac{a.\texttt{alive} = T \qquad [\![e_1]\!]_{\mathbb{E}}(\sigma, a) = T \qquad [\![e_2]\!]_{\mathbb{E}}(\sigma, a) = \lambda \qquad [\![s]\!]_{\mathbb{S}}(\sigma, a) = \mathscr{S}}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a \rangle \rightarrowtail_i \lambda \odot \mathscr{S}}$$

Fig. 6. Semantics for the homogeneous CTMC case. Read as: if the premises on top of the line hold, the conclusion is true.

The operator $\rightarrowtail_r$ that constructs a transition rate for a single rule is defined in (**Exp**). Here, the transition rate function yielded by a whole rule is defined as the pointwise sum of the transition rate functions yielded by all instances of this rule, i.e. all applications of the rule to a single agent. Again, the pointwise addition adds the rates for identical successor states, and keeps distinct successor states separate.

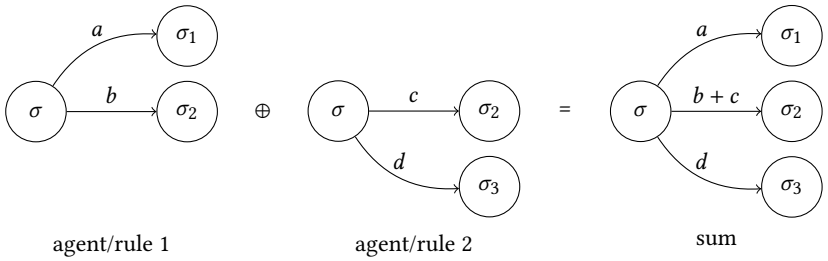

Fig. 7. The effect of the pointwise addition of transition rate functions for two agents/rules.

To define the transition rate function yielded by a single instance, we have to distinguish three cases. In the first case (**Instance-D**), the agent is dead. As only agents that are alive shall exhibit behavior (see Section 3.1.1), these instances yield a constant 0. In the second case (**Instance-F**), the agent is alive, but the guard condition false. Then, the rule will not be applied to this agent and this instance yields a constant 0 as well. In the final case (**Instance-T**), the agent is alive and the guard evaluates to true. In this case the rate expression yields a rate λ, and the effect yields a successor state distribution in the form of a function $\mathscr{S}$ that maps possible successor states to their probabilities. The transition rate for each successor state shall then be the product of its probability and the overall rate, gained by pointwise multiplication of λ and $\mathscr{S}$. As the values of $\mathscr{S}$ add to 1, this conserves the total rate λ.

3.3 Time-dependent transition rates: Non-homogeneous CTMC

While the core-formalism is already sufficient for some models, it does not allow for the modeling of age-dependent (or time-dependent) transitions. As we have seen in the second example (see Figure 2) many demographic processes depend strongly on the age of the involved individuals. Hence, an extension of the formalism is necessary.

While we have based the language without time-dependent transition rates on homogeneous CTMCs, we will now resort to non-homogeneous CTMCs, where transition rates and probabilities may be time-dependent. The procedure stays largely the same, however, the transition rate matrix Q is replaced by a time-dependent matrix $Q(t)$.

We can now define syntax and semantics for the case with time-dependent transitions as we did for the time-independent case. We give a modified definition of the rule syntax in Figure 8. Note that the rule syntax does not change, except that we take rate expression and effect statement from $\mathbb{E}_t$ and $\mathbb{S}_t$ instead of $\mathbb{E}$ and $\mathbb{S}$. Also note that we keep the guard limited to time-independent expressions. While this is not strictly necessary at this point, it will keep things consistent after the introduction of the third language level. This restriction does not impose a loss of expressive power if the expression language includes conditional expressions, as one could make the following transformation without changing semantics:

$$\alpha : e_1 \xrightarrow{e_2} s \leftrightarrow \alpha : \texttt{true} \xrightarrow{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } 0} s \tag{5}$$

As in the time-independent case we define an operator $\rightarrowtail$, using the also defined operators $\rightarrowtail_r$ and $\rightarrowtail_i$. However, the operators' signatures are slightly different, with an additional parameter to account for the time-dependency. The full definition is shown in Figure 9. Note that the only difference to Figure 6 is the introduction of the additional time parameter. Using $\rightarrowtail$ with a specific t we can define the transition matrix $Q(t)$ at time t.

$$r \quad ::= \quad \alpha : e_1 \xrightarrow{e_2} s \qquad \text{(Exp)}$$
$$\textbf{where } \alpha \in \mathcal{A}, e_1 : \texttt{bool} \in \mathbb{E}, e_2 : \texttt{real} \in \mathbb{E}_t, s \in \mathbb{S}_t$$

Fig. 8. Abstract rule syntax with time-dependent rates and effects. Note that we do not allow the guard to be time-dependent.

3.4 Non-stochastic events: GSMP

As we have seen in Section 3, not every demographic process can be appropriately modeled using stochastic rules with exponential distributions. At this third level we introduce new kinds of rules, that are not associated with stochastic waiting times. Figure 10 shows the syntax for those rules. The (Inst) rules shall execute instantly, if the guard is true. Rules described in (Age) shall execute, when the agent reaches a certain age (see Figure 3). And rules described in (Every) shall execute periodically, e.g., for modeling monthly income payment.

As we introduce the new kinds of rules, inter-event times will no longer be exponentially distributed. Hence, the resulting process is no longer a CTMC. Generalized Semi-Markov Processes (GSMP) are a generalization of the CTMC, allowing for arbitrary distributions of event firing times. We will give a short definition and explanation and use that to introduce the notation. Thereby, we mostly follow Glynn [27], but make changes in notation and simplifications whenever it fits our purpose. Most importantly, to simplify equations, we do not include clock evolution rates, as we would set them all to 1.

$$\textbf{(Empty)}\ \frac{}{\langle \epsilon, \sigma, t\rangle \rightarrowtail []} \qquad \textbf{(Rule)}\ \frac{\langle r, \sigma, t\rangle \rightarrowtail_r \mathscr{R} \qquad \langle m, \sigma, t\rangle \rightarrowtail \mathscr{M}}{\langle r\ m, \sigma, t\rangle \rightarrowtail \mathscr{R} \oplus \mathscr{M}}$$

$$\textbf{(Exp)}\ \frac{\sum_{\{(a,\mathscr{I})\ \mid\ a\in\alpha(\sigma), \langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a, t\rangle \rightarrowtail_i \mathscr{I}\}} \mathscr{I} = \mathscr{R}}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, t\rangle \rightarrowtail_r \mathscr{R}}$$

$$\textbf{(Instance-D)}\ \frac{a.\texttt{alive} = F}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a, t\rangle \rightarrowtail_i []} \qquad \textbf{(Instance-F)}\ \frac{a.\texttt{alive} = T \qquad [\![e_1]\!]_{\mathbb{E}}(\sigma, a) = F}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a, t\rangle \rightarrowtail_i []}$$

$$\textbf{(Instance-T)}\ \frac{a.\texttt{alive} = T \qquad [\![e_1]\!]_{\mathbb{E}}(\sigma, a) = T \qquad [\![e_2]\!]_{\mathbb{E}_t}(\sigma, a, t) = \lambda \qquad [\![s]\!]_{\mathbb{S}_t}(\sigma, a, t) = \mathscr{S}}{\langle \alpha : e_1 \xrightarrow{e_2} s, \sigma, a, t\rangle \rightarrowtail_i \lambda \odot \mathscr{S}}$$

Fig. 9. Semantics for the non-homogeneous CTMC case.

$$\begin{array}{lll} r & ::= & \alpha : e_1 \xrightarrow{e_2} s \qquad \text{(Exp)} \\ & \textbf{where} & \alpha \in \mathcal{A}, e_1 : \texttt{bool} \in \mathbb{E}, e_2 : \texttt{real} \in \mathbb{E}_t, s \in \mathbb{S}_t \\ & \mid & \alpha : e_1 \xrightarrow{\texttt{instantly}} s \qquad \text{(Inst)} \\ & \mid & \alpha : e_1 \xrightarrow{\texttt{age}\ e_2} s \qquad \text{(Age)} \\ & \mid & \alpha : e_1 \xrightarrow{\texttt{every}\ e_2} s \qquad \text{(Every)} \\ & \textbf{where} & \alpha \in \mathcal{A}, e_1 : \texttt{bool} \in \mathbb{E}, e_2 : \texttt{real} \in \mathbb{E}, s \in \mathbb{S}_t \end{array}$$

Fig. 10. Abstract rule syntax for level 3. Note that (Exp) is identical to Figure 8, and that in (Age) and (Every) e_2 must not be time-dependent.

Definition 3.2 (Generalized Semi-Markov Process). A Generalized Semi-Markov Process (GSMP) is a tuple $(\mathcal{S}, \sigma_0, \mathcal{E}, E, p, F)$, where:

- $\mathcal{S}$ is a countable set of (physical) states. These correspond to the model states as described above. The *physical* serves to distinguish them from the internal state of the GSMP as described below, which contains some additional information.
- $\sigma_0 \in \mathcal{S}$ is the initial state.
- $\mathcal{E}$ is a countable set of events. With each $\epsilon \in \mathcal{E}$ we will associate a clock $c_\epsilon \in \mathbb{R}$.
- $E : \mathcal{S} \to \mathfrak{p}(\mathcal{E})$ gives the set of active events for each state.
- $p(\sigma'; \sigma, \epsilon, t)$ are the probabilities of transitioning to state σ' from state σ, when the event ϵ happens at time t.
- $F(\cdot; \epsilon, \sigma, t, c_\epsilon)$ is the cumulative distribution function (CDF) of time remaining until the event ϵ triggers. In all cases, $F(\Delta_t; \epsilon, \sigma, t, c_\epsilon)$ must be 0 for all $\Delta_t < 0$, i.e., the probability of the event firing in the past is 0. Note that this differs from the notation in [27]. Our notation is strictly less general, but easier to use for the following definitions.

While the CTMC is defined in terms of transitions between states, a GSMP's evolution is defined in terms of events. Each event may yield different state transitions, given by p, and each state transition may be produced by different events. The CTMC's transition rate matrix Q is replaced by F. Unlike Q, F does not just parameterize one distribution, but it gives a separate distribution for each event.

The internal state of a GSMP is a tuple (σ, t, c), consisting of the (physical) state of the system $\sigma \in \mathcal{S}$, the current time $t \in \mathbb{R}_+$, and a function c that assigns a clock value c_ϵ to each event ϵ. Initially, the process is in the internal state $(\sigma_0, 0, c_0)$, where C_0 has all clocks set to 0.

When the GSMP is in an internal state (σ, t, c), the next internal state is determined as follows: For each event $\epsilon \in E(\sigma)$, a random waiting time Δt_ϵ is drawn from the distribution F. The event ϵ^* with minimal waiting time is selected. If multiple such events exist, one is chosen at random. A new state σ' is assumed at $t' = t + \Delta t_{\epsilon^*}$, drawn according to the state transition probability $p(\sigma', \sigma, \epsilon^*, t')$. Finally, the clocks are updated as follows:

$$c'_\epsilon = -1 \qquad \epsilon \notin E(\sigma') \tag{6}$$

$$c'_\epsilon = 0 \qquad \epsilon \notin E(\sigma) \wedge \epsilon \in E(\sigma') \vee \epsilon = \epsilon^* \tag{7}$$

$$c'_\epsilon = c_\epsilon + \Delta t_{\epsilon^*} \qquad \epsilon \in E(\sigma) \wedge \epsilon \in E(\sigma') \wedge \epsilon \neq \epsilon^* \tag{8}$$

If the event is not active in σ', its clock is set to -1 to mark it as inactive. If the event was previously inactive, but is active in σ', or if the event is the one that was executed, the clock is reset to 0. Otherwise, the clock is advanced by the waiting time of the executed event. This yields the next internal state (σ', t', c').

To give our language a GSMP-semantics we must now define all of the elements of the GSMP $(\mathcal{S}, \sigma_0, \mathcal{E}, E, r, p, F)$ resulting from a model. The set of (physical) states $\mathcal{S}$ remains as defined in Section 3.1.4. Each pair of a possible agent a of type α and an associated rule $\alpha : e_1 \xrightarrow{\cdot} s$ yields an event $\epsilon = (a, \alpha : e \xrightarrow{\cdot} s)$. An event is active, i.e., $\epsilon \in E(\sigma)$, if and only if the agent a exists in this state ($a \in \alpha(\sigma)$), is alive ($a.\texttt{alive} = T$) and the guard is fulfilled ($[\![e_1]\!]_{\mathbb{E}}(\sigma, a) = T$). Transition probabilities are defined as $p(\sigma'; \sigma, \epsilon, t) = [\![s]\!]_{\mathbb{S}_t}(\sigma, a, t)$. We define F separately for each kind of event:

- $\epsilon = (a, \alpha : e_1 \xrightarrow{e_2} s)$: The rule shall behave exactly as defined in Section 3.3, with non-homogeneous CTMC semantics. Hence, the waiting time distribution must be identical to the one generated by the non-homogeneous CTMC, a generalization of the exponential distribution with time-varying parameter.

  $$F(\Delta t; \epsilon, \sigma, t, c_\epsilon) = 1 - e^{-\int_t^{t+\Delta t} [\![e_2]\!]_{\mathbb{E}_t}(\sigma, a, \tau) d\tau}$$

  The distribution is exponential if $[\![e_2]\!]_{\mathbb{E}_t}(\sigma, a, \tau)$ does not depend on τ. Depending on the shape of $[\![e_2]\!]_{\mathbb{E}_t}(\sigma, a, \cdot)$, e.g., if it is only positive on a finite interval, this might not strictly be a CDF, as its limit might be less then 1 (but still non-negative). The difference between 1 and the limit can be interpreted as the probability that the event does not happen at all. It is intuitively clear that this probability must exist, if the rate is only positive on a finite interval.
- $\epsilon = (a, \alpha : e \xrightarrow{\textbf{instantly}} s)$: The event shall be executed instantly, the remaining waiting time is always 0. Hence, the CDF must go from 0 to 1 at $\Delta t = 0$.

  $$F(\Delta t; \epsilon, \sigma, t, c_\epsilon) = \begin{cases} 0, \Delta t < 0 \\ 1, \Delta t \geq 0 \end{cases}$$

- $\epsilon = (a, \alpha : e_1 \xrightarrow{\textbf{age } e_2} s)$: The event shall be executed deterministically, when the agent reaches the age given by the expression e_2. Hence, the CDF goes from 0 to 1 when Δt is the difference between the time when the

agent reaches that age ($[\![e_2]\!]_{\mathbb{E}}(\sigma, a) + a.\texttt{birth}$) and the current time t. Assuming $[\![e_2]\!]_{\mathbb{E}}(\sigma, a) + a.\texttt{birth} \geq t$, i.e., the time when the agent reaches the age given by $[\![e_2]\!]_{\mathbb{E}}(\sigma, a)$, has not yet passed, we define:

$$F(\Delta t; \epsilon, \sigma, t, c_\epsilon) = \begin{cases} 0, \Delta t < [\![e_2]\!]_{\mathbb{E}}(\sigma, a) + a.\texttt{birth} - t \\ 1, \Delta t \geq [\![e_2]\!]_{\mathbb{E}}(\sigma, a) + a.\texttt{birth} - t \end{cases}$$

If that time has passed, no event shall be scheduled at all.

- $\epsilon = (a, \alpha : e_1 \xrightarrow{\textbf{every } e_2} s)$: The event shall be executed periodically, with the period given by e_2. Δt must be the time remaining until the current period is over, while e_ϵ is the time since the event became active or was executed last.

$$F(\Delta t; \epsilon, \sigma, t, c_\epsilon) = \begin{cases} 0, \Delta t < max(0, [\![e_2]\!]_{\mathbb{E}}(\sigma, a) - c_\epsilon) \\ 1, \Delta t \geq max(0, [\![e_2]\!]_{\mathbb{E}}(\sigma, a) - c_\epsilon) \end{cases}$$

The maximum ensures that Δ_t does not become negative when c_ϵ is larger than the period, which might happen as the period may change as it depends on σ.

4 SIMULATION

To apply any simulation modeling language to real-world problems, appropriate efficient simulation algorithms are necessary. Hence, in this section, we will derive a simulation algorithm for each of the three defined language levels.

4.1 Level 1: Exponential Rates

At the first level, we have defined the semantics of the language in terms of Continuous-time Markov Chains (CTMCs). As the CTMC is a common formal basis for modeling formalism, appropriate simulation algorithms are well researched. Fundamentally, most are variations of Gillespie's Stochastic Simulation Algorithm (SSA) [26]. Algorithm 1 lines out one of the simplest variants, the Direct Method, applied to our language. The algorithm samples a single trajectory through the CTMC. Each transition of the CTMC is calculated in two steps. First, the total rate λ for leaving the current state is calculated. Note that the conditions and summations in lines 3 to 8 correspond to the conditions and summations in construction of $\rightarrowtail$ in Figure 6, except that the values for different successor states are also summed. This total rate λ is then used to parameterize the exponential distribution for the time advance (line 11). Second, one of the possible events is selected for execution, whereby each event's probability is given by the proportion of its contribution to the rate sum (line 14). The selected event is then executed. As we allow stochasticity in rule effects, this again involves drawing from a distribution of possible successor states , corresponding to what happens in **(Instance-T)** in Figure 6.

Unfortunately, this algorithm is highly inefficient for large systems. Especially the calculation of λ and the selection of the event will be costly with a large population. However, we can generally expect the effect of events to be local in the social network of the agents. While, for example, the behavior of a single migrant might affect the behavior of his friends and family, other migrants who are socially distant will not be affected. Some variants of the SSA exploit this observation that usually each event will only affect a few other events, e.g. the Next Reaction Method [25] or the Optimized Direct Method [14]. Thereby, a dependency graph is used to keep track of inter-dependencies between events. In the Next Reaction Method, outlined in Algorithm 2 and 3, an event queue is used to select the event that must be executed next. Events are only rescheduled in the event queue after they were executed (line 16) or after an event that changed their rate was executed (lines 19 - 20).

Algorithm 1 Direct Method.
t: current simulation time | t_{end}: end time of the simulation | σ: current simulation state | $\mathcal{A}$: the set of agent types | $\mathcal{R}$: the set of rules | $\lambda_{r,a}$: the rate of rule r applied to agent a | λ: the rate sum

```
1  while t < t_end:
2      // compute rates for all events
3      for each α ∈ A, a ∈ α(σ), r = α : e_1 --e_2--> s ∈ R:
4          if a.alive = T and [[e_1]]_E(σ, a) = T: λ_{r,a} := [[e_2]]_E(σ, a)
5          else: λ_{r,a} := 0
6
7      // compute the rate sum
8      λ := Σ_{α∈A, a∈α(σ), r=α:e_1 --e_2--> s∈R} λ_{r,a}
9
10     // advance simulation time by sampling from an exponential distribution
11     t := t + Exp(λ)
12
13     // select an event to be executed
14     (r, a) := (r*, a*), with P(r*, a*) = λ_{r*,a*} / λ
15
16     // execute the event
17     σ := σ*, with r = α : e_1 --e_2--> s and P(σ*) = ([[s]]_S(σ, a))(σ*)
```

However, this formulation of the algorithm is still vague. Most importantly is unclear, how dependencies will be tracked, and how the dependency graph might look like. In general, the possibilities for dependency tracking depend strongly on the choice of the expression and statement languages $\mathbb{E}$ and $\mathbb{S}$. For the our concrete realization of the language, this procedure is described in detail in [56]. In short, while evaluating a guard or rate, we make note of the agents and attributes that are accessed, and while executing an effect, we make note of which agents and attributes change. This allows us to know which guards and rates might have changed. We have shown that, with a typical model, this exploitation of locality of events leads to improved scalability and therefore significant performance improvements, especially with a larger population of agents.

4.2 Level 2: Time-dependent Rates

For the second language level, we have introduced time-dependent transition rates. In consequence, the resulting CTMC is non-homogeneous, i.e., the transition matrix changes with time. In the abstract, with slight changes, the SSA remains applicable for this class of models [3]. Waiting times between transitions are no longer exponentially distributed, i.e.:

$$F(\Delta t; \lambda) = 1 - e^{-\lambda \Delta t} \tag{9}$$

Instead, they come from a generalized distribution, where the rate parameter λ is a function of time:

$$F(\Delta t; t, \lambda) = 1 - e^{-\int_t^{t+\Delta t} \lambda(\tau) d\tau} \tag{10}$$

In the SSA, drawing from the exponential distribution is replaced by drawing from this distribution. For the selection step of the direct method (Algorithm 1, line 14), the proportions of rates at the time of the event are used.

Algorithm 2 Next Reaction Method.
t: current simulation time | t_{end}: end time of the simulation | σ: current simulation state | $\mathcal{A}$: the set of agent types | $\mathcal{R}$: the set of rules | $\lambda_{r,a}$: the rate of rule r applied to agent a | $\Delta t_{r,a}$: the delay for rule r applied to agent a | Q: the event queue | D: the dependency graph

```
// schedule all potential events in the event queue
for each α ∈ 𝒜, a ∈ α(σ), r = α : e1 --e2--> s ∈ ℛ:
    schedule(r, a)

while t < t_end:
    // select the next event from the queue
    (r, a, Δt) := pop(Q)

    // advance simulation time
    t := t + Δt

    // execute the event
    σ := σ*, with r = α : e1 --e2--> s and P(σ*) = ([[s]]_𝕊(σ, a)) (σ*)

    // reschedule the executed event
    schedule(r, a)

    // reschedule all affected events
    for each (r, a) ∈ affected(D):
        schedule(r, a)
```

Algorithm 3 Next Reaction Method **Schedule**.
(r, a): the event to schedule | t: current simulation time | σ: current simulation state | $\lambda_{r,a}$: the rate of rule r applied to agent a | Q: the event queue | D: the dependency graph

```
function schedule(r, a):
    if a.alive = T and [[e1]]_𝔹(σ, a) = T:
        λ_{r,a} := [[e2]]_𝔹(σ, a)
        Δt_{r,a} ~ Exp(λ_{r,a})
        push(Q, r, a, Δt_{r,a})
    else:
        remove(Q, r, a)
    update(D)
```

While sampling the exponential distribution is trivial, sampling this generalized distribution is challenging. This is especially the case, when the rate can be an arbitrary function of time - as is the case in ML3. For relatively simple shapes of λ the integral can be solved analytically, e.g., linear in time [34] or exponential in time [46]. However, even for only slightly more complex real-world functions, e.g., the Gompertz-Makeham model of mortality with an exponential and a time-independent component [35], analytical solutions are no longer feasible. Numerical solution of the integral is possible, but may be infeasible for performance reasons, as the distribution must be sampled very often: at least

once per event, with additional sampling when events need to be rescheduled. Rejection-based sampling methods, e.g., [60][28], require upper bounds for λ, which can not be determined (or might not even exist) if the expression language $\mathbb{E}_t$ is very expressive. For spline functions, Morgan et al. [50] present a thinning algorithm. Chromar [33] allows to use time-dependent transition rates, but ignores time-dependent changes of the rate during simulation. They assume that the time-dependent dynamics are generally much slower than the event-to-event dynamics of the system, so that the introduced error is small. However, this assumption does not hold for our usual applications, e.g., in the example migration model any agent will typically only be subject to a few dozen events during their lifetime, during which their mortality rate will change drastically. Therefore, in our reference implementation, we restrict the expression language $\mathbb{E}_t$ to only allow piecewise constant functions of time, which we can treat analytically. Other functions must be approximated.

4.3 Level 3: GSMP

At the final, third level, we included additional types of rules that are not governed by exponential rates, but are executed at certain fixed times. With such non-exponential processes we extend beyond the CTMC, and the resulting stochastic process becomes a Generalized Semi-Markov Process (GSMP). In contrast to the CTMC, the GSMP definition is already operational. The fundamental algorithm, similar to the Next Reaction Method for CTMC-based models but with the addition of clocks, is described in the beginning of Section 3.4. For all active events, a waiting time from the respective distributions is drawn. The event with the shortest waiting time is selected, e.g., using an event queue. Then, the selected event is executed. Finally, the clocks are updated. However, in addition to that, each event must be associated with a clock, and clock values must be updated after each event execution, as described in Section 3.4.

Events can be scheduled as in the Next Reaction Method. The condition for an event to be active is identical to the condition used for scheduling in Algorithm 3. Only the drawing from the waiting time distribution (line 3-4) is changed. Instead of the exponential distribution (or the generalized distribution in the non-homogeneous case), is replaced with $F(\mathrm{cot}; \epsilon, \sigma, t, c_\epsilon)$. The tracking of dependencies in the Next Reaction Method, which does not change results, can be included in the same way for the GSMP. The schedule-execute loop remains the same as in Algorithm 2, with the addition of the updating of the clocks' states after line 13. It should be noted that we only need the clock state for the periodic events (**`every`**) - in the three other cases, the waiting time distribution is independent of the clock.

5 CONCRETE LANGUAGE

The above definition of the modeling language is abstract. Not only have we defined the language by relying on some unspecified expression and statement languages ($\mathbb{E}, \mathbb{E}_t, \mathbb{S}, \mathbb{S}_t$), our definition also assumes that the general characteristics of agents and their network are predefined in the set of agent types $\mathcal{A}$ and the set of link definitions $\mathcal{K}$. In this section we describe a concrete realization of the language ML3 as an external domain-specific language, i.e., a new language with syntax independent from (but related to) existing languages. Examples in this section are taken from the model outlined in Section 1. For the rule syntax we refer to the examples in Section 1.

5.1 Agent- and Link-Types

Figure 11 shows the definition of the agent type `Person` (abbreviated to the attributes occurring in the other examples) in the migration model, and the link definition for linking parents and with their children.

The syntax for agent type definitions closely resembles the notation in Section 3.1.1, preceded by the name of the defined type. The required attributes `birth`, `alive` and `id` (see Section 3.1.1) must not be part of the definition. They

```
Person(
    sex: {"m", "f"},
    income: real,
    migrationStage: {"not viable", "intention" , "planning", "preparation", "
        migrated", "exit"},
    migrationStartAge: real,
    ...
)

parents:Person[n] <-> [n]Person:children
```

Fig. 11. Definition of the agent type Person, and of the link between parents and children.

are automatically included. The attribute types of sex and migrationStage are enumeration types, i.e., the attributes may take any of the listed values. The syntax for links uses a different order than in Section 3.1.1. The above definition is equivalent to $\kappa = (children : (Person, \mathsf{n}), parents : (Person, \mathsf{n}))$, which is identical to the example in Figure 4.

5.2 Expression and Statement Language

Our expression language syntactically follows conventions from object-oriented languages. The keyword ego allows access to the agent for which an expression is evaluated. Attributes and link partners may be accessed with a dot notation, e.g., ego.children to get the set of a person's children. In addition to many of the usual arithmetic and logical operations, and a library of predefined commonly used functions, our implementation allows the definition of custom functions (Figure 12). Note that these functions only return a value and can not have side effects, as guard and rate expressions must be evaluated without changing the model state. Model parameters, e.g., rho, alpha, beta and gamma in Figure 12, must be defined in the model. Their values can be passed to the simulator externally.

The time-dependent expression language $\mathbb{E}_t$ is a superset of $\mathbb{E}$ that also allows for accessing an agent's age (e.g., ego.age) and the current time (now).

```
Person.migrationAdvancementRate() : real := rho * e^(ego.migrationIntention())

Person.migrationIntention() : real := alpha * ?MA + beta * ?SN + gamma * ?PBC
where ?MA := ego.migrationAttitude()
?SN := ego.socialNorms()
?PBC := ego.perceivedBehavioralControl()
```

Fig. 12. Definitions of some of the functions used in Section 1. rho, alpha, beta, and gamma are model parameters. A where clause allows the definition of local names. These definitions are evaluated lazily, and only up to once per function call.

The statement language is an imperative language that also follows conventions from object-oriented languages. It allows to assign values and attributes, the definition of variables, setting link partners, creating agents, and the usual control flow constructs, i.e., conditionals and loops. When changing links, the consistency of the changed model state (in regards to the bidirectionality of links) is automatically ensured, as changes are always applied to both directions. When one direction of a link is changed (e.g., a person A is added as a child to a person B), the other direction is

automatically changed accordingly (e.g., the person B is added as the child's parents). When appropriate, statements may contain expressions from an extended version of the expression language. Apart from the aforementioned constructs, expressions used inside a statement may contain stochastic effects, such as selecting a random element from a set (see the fertility rule in Section 1) or drawing from a distribution. Finally, the language allows to define procedures (Figure 13), which unlike the aforementioned functions may change the model state.

```
1 Person.moveToAddress(?address : Address) ->
2 ego.address := ?address
3 ego.friends += ?address.inhabitants + ?address.neighbors.collect(alter.
      inhabitants) - [ego]
```

Fig. 13. Definition of a procedure that moves a migrant to another address. In line 2 its address is changed, and in line 3 other inhabitants and neighbors are added to the migrants social contacts. The predefined function `collect` iterates over a set (here the set of neighbors), evaluates its argument (essentially an anonymous function with parameter `alter`), and collects the results into a single set. As the expressions are set-valued, the plus and minus are set operations, i.e., union and difference.

5.3 Implementation

ML3 is implemented as an interpreted language in Java. A parser implemented with ANTLR 4 parses the model and creates an abstract syntax tree, which is then translated into an internal representation of the model. The latter is then executed by the simulator as described in Section 4. This implementation is available at https://git.informatik.uni-rostock.de/mosi/ml3 under an open source license.

Note that the language, with a syntax that is already very close to many commonly used programming languages, lends itself to a code generation approach, which would promise better performance [42]. Ideally, the target language would be object-oriented, to exploit the close relationship between agent types and classes, and agents and objects. Alternatively, one could implement ML3 as an internal domain specific language, i.e., embedded in a host language [23, pp. 27f.], resulting in a different concrete language that uses host language syntax for the expression and statement languages. A simplified version of ML3, essentially corresponding to the CTMC-based language core with only local interactions between agents, has already been realized as an internal domain specific language embedded in Java [65]. Both code generation as well as implementing ML3 as an internal domain-specific language in Rust are currently pursued.

Simulation experiments with ML3 models are generally performed with SESSL, the Simulation Experiment Specification on a Scala Layer [21]. SESSL is an internal domain-specific language (embedded in the programming language Scala; https://www.scala-lang.org) for specifying and executing various types of simulation experiments. The core of SESSL is simulation-system agnostic, and the connection to different simulation systems is realized via bindings. In this case, we make use of the SESSL-ML3 binding [56].

To resolve dependencies and download software artifacts (e.g., the ML3 simulator), SESSL makes use of Apache Maven (https://maven.apache.org). Usually experiments are distributed in a bundle, containing a Maven wrapper (https://github.com/takari/maven-wrapper) and start scripts, that automatically download and set up Maven, which will then download the necessary software artifacts and execute the experiment. This makes experiments with ML3 models easy to publish and results easy to reproduce.

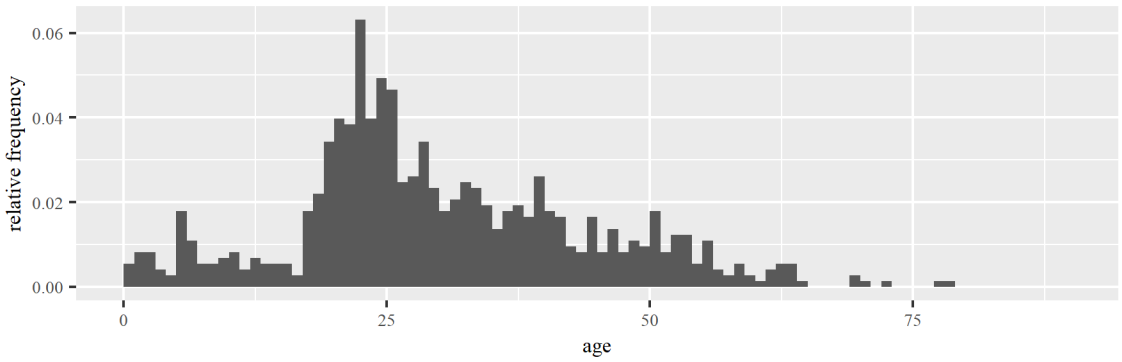

Fig. 14. Distribution of age at migration produced in a single run of the migration decision model.

Figure 14 shows some simulation results, specifically the distribution of the migrant's age at time of migration produced by the migration decision model. To produce the data, the model was initialized with a population of potential migrants sampled from the 1988 Senegal census as detailed by Klabunde et al. in [40], with input data as it is also described there. The software bundle for the presented experiment, including the exact specification of the experimental setup, is available as supplementary material. Further examples of models in ML3 can be found in [55, 57].

6 RELATED WORK

We relate ML3 to other approaches for formally defining languages for modeling and simulation. For a more hands-on comparison see our earlier review [59].

To describe agent-based models in a readable and executable manner, a number of popular frameworks and languages like Repast Simphony [52], NetLogo [67], or Mesa [48] are available. However, these languages are largely limited to discrete-time (or "stepwise") simulation rather than more general discrete-event simulation in continuous time, and approaches with fixed time-steps prevail. Law states that "[t]he primary uses of [fixed-increment time advance] appear to be for systems where it can reasonably be assumed that all events actually occur at one of the times $n\Delta t$ ($n = 0, 1, 2, \ldots$) for an appropriately chosen Δt and, unfortunately, for agent-based simulation" [45, p. 73]. In comparison to discrete-event simulation in continuous time, discrete-time simulation includes two sources of error [45, section 1A]. First, the selected time step in addition with the updating scheme, for example whether all agents are updated/interact synchronously or asychronously, might introduce some significant bias into the simulation results. E.g., Özmen et al. [53] compared models with different update schemes of the 1918 influenza pandemic, and found significant differences in peak load and diffusion speed depending on the update scheme.

Second, the exact timing of events and, consequently, the time between events is lost in discrete-time simulation. In the context of demography, Willekens [68] argues that continuous time would allow the modeler to define a sequence of events (transitions) rather than the state occupancies at successive points in time. This allows a precise measurement of the lengths of episodes between events, whereas in discrete-time simulation these lengths could only be approximated. Finally, the problem "how to handle multiple transitions during a same interval"[68, p. 354] disappears.

Many formal languages for the modeling and simulation of stochastic processes are based on the rule-based paradigm. In particular for biological application, rule-based modeling reflects domain-specific concepts like biochemical reactions.

Influential languages include Kappa [16] and BioNetGen [7]. Whereas biological rule-based modeling languages and ML3 share the idea of rules governing the behavior of the system, they take different perspectives. Rule-based modeling languages focus on describing reactions, with the entities taking a passive role. In contrast, ML3 attaches the behavior rules to the entities, making them the active elements of the system. This duality of "reaction systems" and "reactive systems" is well known [22, 62, 63].

One prominent example for the reactive systems metaphor is DEVS [69], in which encapsulated entities communicate via strictly defined interfaces. DEVS structures entities in a tree, allowing hierarchical composition of models. Cell-DEVS has recently been applied to model social phenomena as discrete-event cellular automata[39]. DEVS Markov [58] is a variant of DEVS with Markovian semantics, including both discrete-time and continuous time Markov processes. In any of these variations, DEVS' virtue is the support of modular hierarchical modeling, which is supported by the formalism being closed under coupling. However, flat, graph-like model structures are better suited for demographic ABMs. Therefore, ML3 uses a very general notion of links to facilitate the representation of linked lives, rather than DEVS's strict hierarchies. In DEVS the interaction (coupling) information is kept at the next higher level and the entities themselves do not have access to the information with whom they are interacting. In contrasst, ML3's link information is stored with the agents, and agents directly know the entities involved in an interaction. As links, and with them interactions, can change and agents can be created in ML3, variable structures, which form a salient feature of agent-based simulation, are supported [61].

More close relatives of ML3 are process algebras. In particular the attributed π-calculus [37], which augments processes with local attributes, shares some similarities with ML3. Here, whether a specific behavior occurs and how likely (or, equivalently, how fast) it is, can depend on the local state of a process. Also related are process calculi and languages in the context of Collective Adaptive Systems, e.g., AbC [2], CARMA [9]. For CARMA, the CARMA Specification Language [31] was developed as a higher-level language for model implementation. In contrast, we developed the language first, in close collaboration with modelers from the domain to adapt it to their requirements, and formalized it later.

The change of interactions or communication patterns as a salient process algebra and ML3 share, this is in contrast to other formalisms such as Petri Nets or DEVS, where the changes of variable interactions or compositions

The distinctive difference between ML3 and process algebras is the way the effect of a specific behavior is expressed. Process algebras rely on replacing a process definition with another one, potentially recursively. In contrast to this declarative approach, ML3 represents reaction effects as imperative mutations of the model state. In particular for expressing changes of network links, imperative updates provide an often succinct alternative to declarative specifications.

Although most formal modeling languages follow a declarative approach, some modeling languages with imperative elements have been proposed. For example, the imperative π-calculus [36] extends the attributed π-calculus with imperative statements for updating a global key-value store. In the modeling language ℓ [70], biochemical reaction rules are combined with imperative blocks. More similar to ML3 are languages that employ stochastic guarded commands [30], such as SABRE [19] or the input language for the PRISM model checker [44]. In PRISM, stochastic guarded commands are used to describe diverse stochastic process models. This shows the expressive power of the approach. In contrast to these existing languages, ML3 exploits stochastic guarded commands to represent changes in networks of attributed agents.

Considering ML3's focus on graph structures brings us back to rule-based modeling languages for biochemistry. Kappa [16] and BioNetGen [7], for example, consider the reactants of a rule as a graph pattern. The root of these modeling languages lies in the Bigraph formalisms [49], which has been extended to stochastic Bigraphs [43]. These

approaches use graphs and graph patterns to express structural changes in molecules. However, the individual graph vertices and their dynamics are less complex than in ML3. In particular, ML3 allows attributes of arbitrary types for each agent, whereas languages for biochemistry typically are aimed at encoding entities with a comparatively small number of attributes with few possible values, such as being phosphorylated or not. Effects in ML3 may be arbitrary imperative code applied to the network of agents with their current state. Especially the latter makes the encoding of ML3-rule effects as replacement of graph patterns infeasible.

Glynn [27] proposed the GSMP as a general formal framework of discrete event systems and for many formalisms variants with semi-markovian semantics have been proposed, to be applied when exponential delays are inappropriate or insufficient. Examples include Deterministic and Stochastic Petri Nets [47], which resemble ML3 in that they combine exponential delays with deterministically scheduled transitions, and stochastic process algebras, e.g., GSMPA [11] and IGSMP [12]. GSMPs have also been used to describe CTMCs with added deterministic delays [10, 15]. These delayed transitions yield two state changes. The first one timed by an exponential distribution, and the second follows after a deterministic delay. In the GSMP this results in two events: one with an exponential distribution, and a second one, which only becomes active after the first one was executed, that is deterministically timed. This is similar to the periodic events in ML3, which are delayed after their previous occurrence, while ML3's **age**-events are deterministically timed relative to the simulation time t, not to previous events.

In summary, ML3's novelty lies in its combination of established concepts. ML3 merges graph-based formalisms with an individual-centric perspective and employs stochastic guarded commands to describe changes succinctly.

7 DISCUSSION

Unlike many of the formalisms listed above, ML3 was developed as a modeling language first, and formalized later. Hence, at the design phase, the focus was on the requirements of the application domain and the needs of the modeler (see the considerations in [66], where an earlier version of the language is presented). While the Markovian semantics of the language's core (see Section 3.2) was roughly outlined early on, and the implementation of the simulation software served as somewhat of a formalization, the firm grounding of the language in precisely defined stochastic processes presented in this paper occurred much later. Having defined the formal semantics, we use it as a starting point to discuss some properties of using ML3 for specifying simulation models.

First, the formal semantics of ML3 states that an ML3 model is interpreted as a stochastic process in continuous time. As described in Section 4, the corresponding simulation algorithm relies on scheduling and executing discrete events at arbitrary time points. Thus, in contrast to discrete-time simulation approaches, ML3 allows an *exact* modeling of continuous-time processes. For example, the linked lives model by Noble et al. [51] uses a discrete time step of one year. All state changes are executed at the end of each year in some chosen order, causing artifacts in the model. For example, the model first decides whether a person dies in a year before considering other behaviors. Therefore, in that model a person can not get married and then die during one year. With ML3, events occur in continuous time and, thus, are naturally ordered, see [66] for a more thorough discussion and an ML3 implementation of the linked lives model.

Second, the formal semantics precisely describes that a model is built from rules in a compositional manner. The overall behavior of the model is obtained by evaluating each rule for each agent independently. From the resulting waiting times, the shortest one is selected for execution. Thus, competing stochastic behaviors (for example, marriage and death as above) are interpreted as independently running in parallel in a stochastic race. This is parallel composition in process algebras [18]. Apart from this stochastic race, rules never interact with each other. Therefore, rules can be

reasoned about individually, and removing a rule from or adding a rule to a well-defined model will always yield a well-defined model. This facilitates reading and editing ML3 models.

Third, the formal semantics describes a precise interface between the model syntax and its behavior. The separation between syntax and semantics can be adopted in the implementation of the language. This leads directly to the separation of model and simulator, where the model is written in a concrete syntax, parsed into the abstract syntax, which is then used as input for a simulator (see Section 5.3). As a consequence, the simulation algorithm is reusable across models. Moreover, the models do not contain any execution-specific aspects, making them much more succinct and readable than implementations that mix behavior and execution code [65].

While a variety of modeling languages and formalisms are based on stochastic events with exponential delays, including many of those listed in Section 6, some of the features of ML3 are more unusual and warrant further discussion.

First, ML3 supports time-dependent transition rates. While the formalization of time-dependent rates with non-homogeneous CTMCs is relatively straight forward, having time-dependent expressions available raises the question where else they might be used outside the rates. That the effects of rules must be time-dependent as well follows immediately - just consider an event, where a new agent is created, i.e., a birth. If the agents' age, i.e., the time of their birth is tracked, the exact state change is already dependent on the event time. In general time-dependent effects are not problematic - when the event is executed, its execution time is of course well known. More care had to be taken with the expressions involved with the scheduled events with **`age`** and **`every`**. It would not be immediately clear what a clause such as `@ every ego.age` would mean. To avoid complications, and potential confusions for the modeler, we disallow time-dependency in these places.

The waiting time distribution yielded by time-dependent transition rates is much more difficult to sample than the exponential distribution, which is the waiting-time distribution in the time-independent case. So while the formalization as a CTMC is very similar to the homogeneous case without time-dependent rates, the non-homogeneous case has an additional challenge for execution. While age-dependent behavior is central life-course models that motivated the language (e.g, [41, 51]), other models from demography (or the broader social sciences) do not require this (e.g., the model in [55]). Given the additional challenge for simulation introduced in the non-homogeneous case, and the existence of models not requiring that, this non-homogeneous core is an interesting and relevant subset of the language. The separate formalization of this core enables us to investigate it further in future work, e.g., by implementing specialized simulators that may make additional optimizations that might be impossible in the non-homogeneous case.

Originally, periodic events (with the keyword **`every`**) would be scheduled to the whole multiples of the period length (e.g., at $t = \frac{1}{12}, \frac{2}{12}, \frac{3}{12}, \ldots$ if the period was $\frac{1}{12}$), and different events of different agents with the same period would be synchronized. This was motivated by the original use case: agents should get monthly income payments, which should happen at the same time for everyone. When formalizing the language, it became apparent that this mechanism is actually relatively complicated to define, and to explain. From a purely modeling perspective it is not clear that different agents should synchronize their behavior. Hence, we changed the mechanism to the one presented in Section 3.4. In this version - which also fits much better into the GSMP formalization - agents no longer act synchronously without apparent reason, as each operates with their own clock. Synchronous behavior as described above can still be implemented by having it as behavior of a single central agent, e.g., the source of the income payments. This includes the reason of the synchronicity into the model. Both these examples show benefits of formalization for language design.

The inclusion of events with non-stochastic timing also reintroduces some of the issues related to discrete-time simulation. In particular, multiple events may again be scheduled to happen at the same time, and need to be ordered for execution. However, unlike many discrete time models, we define how the ordering shall occur (Section 3.4),

i.e., randomly. In combination with the very powerful language for rule effects, ML3 allows, as the most extreme example, to implement a completely step-wise model using **every**. Using this feature carefully and sparingly remains the responsibility of the user. The formal definition of the language at different levels, with non-stochastic events not being part of the core language, shall not only make the relationship to the more common CTMC approaches clear, but also reinforce the point that non-stochastic events should not be regarded as the default option, but only be used after careful consideration.

```
Person
|  ego.inMigrationProcess(), ego.migrationIntention() >= 0, ego.migrationStage
     = "preparation", ego.canAffordMigration()
@  ego.migrationAdvancementRate() * ego.successProbability()
-> ego.migrate()

Person
|  ego.inMigrationProcess(), ego.migrationIntention() >= 0, ego.migrationStage
     = "preparation", ego.canAffordMigration()
@  ego.migrationAdvancementRate() * (1 - ego.successProbability())
-> ego.failMigration()
```

Fig. 15. Alternative formulation of the behavior implemented as one rule in Figure 1.

Apart from such relatively clear misuses of the language, the relative flexibility of ML3 also leads to situations where there is more than one way to implement a given process. For example, migration behavior shown in Figure 1 might, with identical CTMC semantics, be implemented differently, with two rules (see Figure 15). This results from the fact that rule effects can be stochastic, and the language allows a stochastic conditional in the effect. However. while semantically identical, from a modelers perspective these two variants might be very different. In the version with one rule, the rule represents the migrant attempting to migrate - which can have two different outcomes. With two rules, these two outcomes become two different behavioral options. In this case, the version with one rule seems more fitting from a modeling perspective. It is the responsibility of the modeler to choose the most suitable alternative. The formal semantics enables formal reasoning about rule equivalence in future work.

8 CONCLUSION

In this paper we retrofit a formal semantics to ML3, a modeling language for agent-based discrete-event simulation in demography. The characteristics of the application domain demography are reflected in the language and its formal semantics. In particular, age-dependent behavior and non-stochastic events can not be expressed with homogeneous Continuous-Time Markov Chains, which are the most common semantic domain for formal stochastic simulation modeling languages [8]. Instead, the formal semantics of ML3 relies on Generalized Semi-Markov Processes. These provide the expressiveness necessary to capture essential processes in agent-based discrete-event models in demography. In addition, the design supports the compositional development and succinct description of simulation models.

Equipping such a simulation modeling language with a formal semantics facilitates an unambiguous interpretation of model definitions. An ML3 model is mapped to a stochastic process by the formal semantics, allowing formal reasoning about the model. Competing behaviors are encoded as parallel rules, and the stochastic race between them is precisely defined in the semantics. This is fundamentally different than *ad hoc* implementations of models

in general-purpose programming languages, which fuse code for model behavior and simulation algorithm. Being a domain-specific language for simulation, ML3 clearly separates model and simulator. In addition, implementations in general-purpose programming languages often rely on discrete-time simulation and contain "hidden" assumptions about the synchronization and ordering of simultaneous events. In contrast, in ML3 events are scheduled in continuous time avoiding these problems.

Currently, ML3 is implemented as an external DSL. However, ML3 shares many concepts with object-oriented programming languages, particularly for expressions and statements about agents. In the future we will explore ways to implement ML3 as an internal DSL with an object-oriented host language, which should allow us to reuse syntax and semantics of the host language. This will simplify the tooling around the language. In addition, users will be able to transfer their knowledge of an existing language to ML3. Furthermore, the formal semantics will inform the implementation of new, sound and more efficient, simulation algorithms.

9 ACKNOWLEDGMENTS

This research received funding from the European Research Council (ERC) via the grant Bayesian Agent-based Population Studies (CoG-2016-725232) and the German Research Foundation (DFG) via the grant Modeling and Simulation of Linked Lives in Demography (UH–66/15). We thank Anna Klabunde and Frans Willekens for the many discussions that lead to the design of the language and its usability. We thank Jakub Bijak for his comments on the manuscript.

REFERENCES

[1] Icek Ajzen. 1991. The Theory of Planned Behavior. *Organizational Behavior and Human Decision Processes* 50, 2 (1991), 179–211. https://doi.org/10.1016/0749-5978(91)90020-T

[2] Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. 2016. On the Power of Attribute-Based Communication. In *Formal Techniques for Distributed Objects, Components, and Systems (Lecture Notes in Computer Science)*. Springer, Cham, 1–18.

[3] David F. Anderson. 2007. A Modified next Reaction Method for Simulating Chemical Systems with Time Dependent Propensities and Delays. *The Journal of Chemical Physics* 127, 21 (2007), 214107. https://doi.org/10.1063/1.2799998

[4] Belinda Aparicio Diaz, Thomas Fent, Alexia Prskawetz, and Laura Bernardi. 2011. Transition to Parenthood: The Role of Social Interaction and Endogenous Networks. *Demography* 48, 2 (2011), 559–79.

[5] Francesco Billari, Belinda Aparicio Diaz, Thomas Fent, and Alexia Fürnkranz-Prskawetz. 2007. The "Wedding-Ring": An Agent-Based Marriage Model Based on Social Interaction. *Demographic Research* 17, 3 (2007), 59–82. https://doi.org/10.4054/DemRes.2007.17.3

[6] Francesco C. Billari and Alexia Prskawetz (Eds.). 2003. *Agent-Based Computational Demography: Using Simulation to Improve Our Understanding of Demographic Behaviour*. Physica-Verlag Heidelberg. https://doi.org/10.1007/978-3-7908-2715-6

[7] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. 2004. BioNetGen: Software for Rule-based Modeling of Signal Transduction Based on the Interactions of Molecular Domains. *Bioinformatics* 20, 17 (2004), 3289–3291.

[8] Henrik C. Bohnenkamp and Boudewijn R. Haverkort. 1999. Semi-numerical Solution of Stochastic Process Algebra Models. In *Formal Methods for Real-Time and Probabilistic Systems*, Joost-Pieter Katoen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 228–243.

[9] Luca Bortolussi, Rocco De Nicola, Vashti Galpin, Stephen Gilmore, Jane Hillston, Diego Latella, Michele Loreti, and Mieke Massink. 2015. CARMA: Collective Adaptive Resource-Sharing Markovian Agents. *Electronic Proceedings in Theoretical Computer Science* 194 (2015), 16–31.

[10] Luca Bortolussi and Jane Hillston. 2012. Fluid approximation of CTMC with deterministic delays. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*. IEEE, 53–62.

[11] Mario Bravetti, Marco Bernardo, and Roberto Gorrieri. 1997. From EMPA to GSMPA: Allowing for General Distributions. In *Proceeding of the 5th International Workshop on Process Algebras and Performance Modeling*.

[12] Mario Bravetti and Roberto Gorrieri. 2002. The Theory of Interactive Generalized Semi-Markov Processes. *Theoretical Computer Science* 282, 1 (2002), 5–32. https://doi.org/10.1016/S0304-3975(01)00043-3

[13] Pierre Brémaud. 1999. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Number 31 in Texts in Applied Mathematics. Springer, New York.

[14] Yang Cao, Hong Li, and Linda Petzold. 2004. Efficient Formulation of the Stochastic Simulation Algorithm for Chemically Reacting Systems. *The Journal of Chemical Physics* 121, 9 (2004), 4059–4067. https://doi.org/10.1063/1.1778376

[15] Giulio Caravagna and Jane Hillston. 2012. Bio-PEPAd: A Non-Markovian Extension of Bio-PEPA. *Theoretical Computer Science* 419 (Feb. 2012), 26–49. https://doi.org/10.1016/j.tcs.2011.11.028

[16] Vincent Danos and Cosimo Laneve. 2004. Formal molecular biology. *Theoretical Computer Science* 325, 1 (2004), 69–110.

[17] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. 2009. On a Uniform Framework for the Definition of Stochastic Process Languages. In *Formal Methods for Industrial Critical Systems (Lecture Notes in Computer Science)*, María Alpuente, Byron Cook, and Christophe Joubert (Eds.). Springer, Berlin, Heidelberg, 9–25.

[18] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. 2013. A Uniform Definition of Stochastic Process Calculi. *Comput. Surveys* 46, 1 (2013), 5:1–5:35.

[19] Frederic Didier, Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. 2010. SABRE: A Tool for Stochastic Analysis of Biochemical Reaction Networks. In *Proceedings of the 7th International Conference on the Quantitative Evaluation of Systems*. 193–194. https://doi.org/10.1109/QEST.2010.33

[20] Joshua M. Epstein and Robert Axtell. 1996. *Growing Artificial Societies: Social Science from the Bottom Up.* Brookings Institution Press.

[21] Roland Ewald and Adelinde M. Uhrmacher. 2014. SESSL: A Domain-Specific Language for Simulation Experiments. *ACM Transactions on Modeling and Computer Simulation* 24, 2 (2014), 11:1–11:25.

[22] Jasmin Fisher, David Harel, and Thomas A. Henzinger. 2011. Biology as Reactivity. *Commun. ACM* 54, 10 (2011), 72–82. https://doi.org/10.1145/2001269.2001289

[23] Martin Fowler. 2010. *Domain-specific languages.* Pearson Education.

[24] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. 1993. Directed Hypergraphs and Applications. *Discrete Applied Mathematics* 42, 2 (1993), 177–201. https://doi.org/10.1016/0166-218X(93)90045-P

[25] Michael A. Gibson and Jehoshua Bruck. 2000. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *The Journal of Physical Chemistry A* 104, 9 (2000), 1876–1889. https://doi.org/10.1021/jp993732q

[26] Daniel T. Gillespie. 1977. Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry* 81, 25 (1977), 2340–2361. https://doi.org/10.1021/j100540a008

[27] P.W. Glynn. 1989. A GSMP Formalism for Discrete Event Systems. *Proc. IEEE* 77, 1 (1989), 14–23. https://doi.org/10.1109/5.21067

[28] Gerrit Großmann, Luca Bortolussi, and Verena Wolf. 2020. Rejection-Based Simulation of Non-Markovian Agents on Complex Networks. In *Complex Networks and Their Applications VIII (Studies in Computational Intelligence)*, Hocine Cherifi, Sabrina Gaito, José Fernendo Mendes, Esteban Moro, and Luis Mateus Rocha (Eds.). Springer International Publishing, Cham, 349–361.

[29] Alexander Helleboogh, Giuseppe Vizzari, Adelinde M. Uhrmacher, and Fabien Michel. 2007. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems* 14, 1 (2007), 87–116. http://eprints.mosi.informatik.uni-rostock.de/109/

[30] Thomas A. Henzinger, Barbara Jobstmann, and Verena Wolf. 2011. Formalisms for Specifying Markovian Population Models. *International Journal of Foundations of Computer Science* 22, 4 (2011), 823–841.

[31] Jane Hillston and Michele Loreti. 2016. CARMA Eclipse Plug-in: A Tool Supporting Design and Analysis of Collective Adaptive Systems. In *Quantitative Evaluation of Systems (Lecture Notes in Computer Science)*. Springer, Cham, 167–171.

[32] Martin Hinsch and Jakub Bijak. 2019. Rumours Lead to Self-Organized Migration Routes. In *The 2019 Conference on Artificial Life: How Can Artificial Life Help Solve Societal Challenges?*

[33] Ricardo Honorato-Zimmer, Andrew J. Millar, Gordon D. Plotkin, and Argyris Zardilis. 2017. Chromar, a Language of Parameterised Agents. *Theoretical Computer Science* (2017). https://doi.org/10.1016/j.tcs.2017.07.034

[34] A. P. J. Jansen. 1995. Monte Carlo Simulations of Chemical Reactions on a Surface with Time-Dependent Reaction-Rate Constants. *Computer Physics Communications* 86, 1 (1995), 1–12. https://doi.org/10.1016/0010-4655(94)00155-U

[35] P. Jodrá. 2009. A Closed-Form Expression for the Quantile Function of the Gompertz–Makeham Distribution. *Mathematics and Computers in Simulation* 79, 10 (2009), 3069–3075. https://doi.org/10.1016/j.matcom.2009.02.002

[36] Mathias John, Cédric Lhoussaine, and Joachim Niehren. 2009. Dynamic Compartments in the Imperative π-Calculus. In *Computational Methods in Systems Biology*, Pierpaolo Degano and Roberto Gorrieri (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 235–250.

[37] Mathias John, Cédric Lhoussaine, Joachim Niehren, and Adelinde M. Uhrmacher. 2008. The Attributed Pi Calculus. In *Computational Methods in Systems Biology*. Springer-Verlag, Berlin, Heidelberg, 83–102.

[38] Ridhi Kashyap and Francisco Villavicencio. 2016. The Dynamics of Son Preference, Technology Diffusion, and Fertility Decline Underlying Distorted Sex Ratios at Birth: A Simulation Approach. *Demography* 53 (2016), 1261–1281.

[39] Hoda Khalil and Gabriel Wainer. 2020. Cell-DEVS for Social Phenomena Modeling. *IEEE Transactions on Computational Social Systems* 7, 3 (2020), 725–740. https://doi.org/10.1109/TCSS.2020.2982885

[40] A. Klabunde, 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.* Max Planck Institute for Demographic Research.

[41] Anna Klabunde, Sabine Zinn, Frans Willekens, and Matthias Leuchter. 2017. Multistate Modelling Extended by Behavioural Rules: An Application to Migration. *Population Studies* 71, sup1 (2017), 51–67.

[42] Till Köster, Tom Warnke, and Adelinde M. Uhrmacher. 2020. Partial Evaluation via Code Generation for Static Stochastic Reaction Network Models. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '20)*. Association for Computing Machinery, Miami, FL, Spain, 159–170. https://doi.org/10.1145/3384441.3395983

[43] Jean Krivine, Robin Milner, and Angelo Troina. 2008. Stochastic Bigraphs. *Electronic Notes in Theoretical Computer Science* 218 (2008), 73–96. https://doi.org/10.1016/j.entcs.2008.10.006

[44] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 585–591.

[45] Averill M. Law. 2014. *Simulation Modeling and Analysis* (5th ed.). McGraw-Hill Education, Dubuque.

[46] T. Lu, D. Volfson, L. Tsimring, and J. Hasty. 2004. Cellular Growth and Division in the Gillespie Algorithm. *Systems Biology* 1, 1 (2004), 121–128. https://doi.org/10.1049/sb:20045016

[47] M. Ajmone Marsan and G. Chiola. 1987. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. In *Advances in Petri Nets 1987 (Lecture Notes in Computer Science)*, Grzegorz Rozenberg (Ed.). Springer Berlin Heidelberg, 132–145.

[48] David Masad and Jacqueline L. Kazil. 2015. Mesa: An Agent-Based Modeling Framework. In *Proceedings of the 14th Python in Science Conference*. https://doi.org/10.25080/majora-7b98e3ed-009

[49] Robin Milner. 2001. Bigraphical Reactive Systems. In *Proceedings of the 12th International Conference on Concurrency Theory*. Springer-Verlag, Berlin, Heidelberg, 16–35.

[50] Lucy E. Morgan, Barry L. Nelson, Andrew C. Titman, and David J. Worthington. 2019. A Spline-Based Method for Modelling and Generating A Nonhomogeneous Poisson Process. In *Proceedings of the 2019 Winter Simulation Conference*. IEEE Press, 356–367. https://doi.org/10.1109/WSC40007.2019.9004867

[51] Jason Noble, Eric Silverman, Jakub Bijak, Stuart Rossiter, Maria Evandrou, Seth Bullock, Athina Vlachantoni, and Jane Falkingham. 2012. Linked Lives: The Utility of an Agent-Based Approach to Modeling Partnership and Household Formation in the Context of Social Care. In *Proceedings of the 2012 Winter Simulation Conference*. IEEE Press, 93:1–93:12.

[52] Michael J North, Nicholson T Collier, Jonathan Ozik, Eric R Tatara, Charles M Macal, Mark Bragen, and Pam Sydelko. 2013. Complex Adaptive Systems Modeling with Repast Simphony. *Complex Adaptive Systems Modeling* 1, 1 (2013), 3. https://doi.org/10.1186/2194-3206-1-3

[53] Özgür Özmen, James J Nutaro, Laura L Pullum, and Arvind Ramanathan. 2016. Analyzing the impact of modeling choices and assumptions in compartmental epidemiological models. *Simulation* 92, 5 (2016), 459–472.

[54] Benjamin C. Pierce. 2002. *Types and programming languages*. MIT Press, Cambridge, MA, USA.

[55] Oliver Reinhardt, Jason D. Hilton, Tom Warnke, Jakub Bijak, and Adelinde M. Uhrmacher. 2018. Streamlining Simulation Experiments with Agent-Based Models in Demography. *Journal of Artificial Societies and Social Simulation* 21, 3 (2018), 9.

[56] Oliver Reinhardt and Adelinde M. Uhrmacher. 2017. An Efficient Simulation Algorithm for Continuous-Time Agent-Based Linked Lives Models. In *Proceedings of the 50th Annual Simulation Symposium*. Society for Computer Simulation International, 9:1–9:12.

[57] Oliver Reinhardt, Adelinde M. Uhrmacher, Martin Hinsch, and Jakub Bijak. 2019. Developing Agent-Based Migration Models in Pairs. In *Proceedings of the 2019 Winter Simulation Conference*, N. Mustafee, K.-H. G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son (Eds.). IEEE, Piscataway, New Jersey, 2713–2724. https://doi.org/10.1109/WSC40007.2019.9004946

[58] Chungman Seo, Bernard P. Zeigler, and Doohwan Kim. 2018. DEVS Markov Modeling and Simulation: Formal Definition and Implementation. In *Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences (ICCES'18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3213187.3213188

[59] A. Steiniger, A. M. Uhrmacher, S. Zinn, J. Gampe, and F. Willekens. 2014. The role of languages for modeling and simulating continuous-time multi-level models in demography. In *Proceedings of the Winter Simulation Conference 2014*. IEEE Press, 2978–2989.

[60] Vo Hong Thanh and Corrado Priami. 2015. Simulation of Biochemical Reactions with Time-Dependent Rates by the Rejection-Based Algorithm. *The Journal of Chemical Physics* 143, 5 (2015), 054104. https://doi.org/10.1063/1.4927916

[61] Adelinde Uhrmacher. 1996. Object-oriented and agent-oriented simulation: Implications for social science application. In *Social Science Microsimulation*. Springer, 432–447.

[62] Adelinde M. Uhrmacher. 1997. Concepts of Object- and Agent-Oriented Simulation. *Transactions of the Society for Computer Simulation International* 14, 2 (1997), 59–67.

[63] A. M. Uhrmacher. 2001. Dynamic Structures in Modeling and Simulation: A Reflective Approach. *ACM Transactions on Modeling and Computer Simulation* 11, 2 (2001), 206–232. https://doi.org/10.1145/384169.384173

[64] Tom Warnke, Oliver Reinhardt, Anna Klabunde, Frans Willekens, and Adelinde M. Uhrmacher. 2017. Modelling and Simulating Decision Processes of Linked Lives: An Approach Based on Concurrent Processes and Stochastic Race. *Population Studies* 71, sup1 (2017), 69–83.

[65] Tom Warnke, Oliver Reinhardt, and Adelinde M. Uhrmacher. 2016. Population-Based CTMCs and Agent-Based Models. In *Proceedings of the 2016 Winter Simulation Conference (WSC '16)*. IEEE Press, 1253–1264.

[66] Tom Warnke, Alexander Steiniger, Adelinde M. Uhrmacher, Anna Klabunde, and Frans Willekens. 2015. ML3: A Language for Compact Modeling of Linked Lives in Computational Demography. In *Proceedings of the 2015 Winter Simulation Conference (WSC '15)*. IEEE Press, 2764–2775.

[67] Uri Wilensky. 1999. NetLogo. https://ccl.northwestern.edu/netlogo/.

[68] Frans Willekens. 2009. Continuous-Time Microsimulation in Longitudinal Analysis. In *New Frontiers in Microsimulation Modelling*, A. Zaidi, A. Harding, and P. Williamson (Eds.). Ashgate, 413–436.

[69] Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. 2018. *Theory of modeling and simulation: Discrete event & iterative system computational foundations* (3rd ed.). Academic Press, Inc.

[70] Roberto Zunino, Đurica Nikolić, Corrado Priami, Ozan Kahramanoğulları, and Tommaso Schiavinotto. 2015. ℓ: An Imperative DSL to Stochastically Simulate Biological Systems. In *Programming Languages with Applications to Biology and Security*, Chiara Bodei, Gian-Luigi Ferrari, and Corrado Priami (Eds.). Vol. 9465. Springer International Publishing, Cham, 354–374.