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.

DEVELOPING AGENT-BASED MIGRATION MODELS IN PAIRS

Oliver Reinhardt
Adelinde M. Uhrmacher

Visual and Analytic Computing
University of Rostock
Albert-Einstein-Straße 22
18059 Rostock, GERMANY

Martin Hinsch
Jakub Bijak

Social Statistics and Demography
University of Southampton
University Road
Southampton SO17 1BJ, UK

ABSTRACT

Developing a realistic agent-based model of human migration requires particular care. Committing too early to a specific model architecture, design, or language environment can later become costly in terms of the revisions required. To examine specifically the impact of differences in implementation, we have developed two instances of the same model in parallel. One model is realized in the programming language Julia, the underlying execution semantics is of a discrete stepwise stochastic process. The other is realized in an external domain-specific language ML3, based on a continuous-time Markov chain (CTMC) semantics. By developing models in pairs in different approaches, important properties of the target model can be more effectively revealed. In addition, the realization within a programming language and an external domain-specific modeling language respectively, helped identifying crucial features and trade-offs for the future implementation of the model and the design of the domain-specific modeling language.

1 INTRODUCTION

Human migration is an ubiquitous, as well as fundamentally important demographic phenomenon. Not only can migration have substantial cultural, political, economic and demographic effects in all countries involved (i.e. origin, transit and destination), but it has also always been an important and contentious political topic (with new, increased relevance in recent years, see Leurs and Smets 2018; Ekman 2018). Still, for all its importance, migration remains a very complex phenomenon, and one of the most uncertain demographic processes, largely evading theoretical attempts to explain it. While there is a body of theory on migration based on a top-down, economic paradigm, comparatively few attempts have been made to understand migration from the bottom up (Massey et al. 1993).

In this work, we are interested in explaining a small part of the migration picture, which is how availability and transfer of information affect the establishment and optimality of migration routes in a forced migration scenario. During their journey, migrants seeking asylum in safe third countries act under strong constraints concerning the availability of resources and information (Borkert et al. 2018). It appears that most individuals base their decisions on information they have received from social contacts or “unofficial” (as opposed to state- or NGO-sponsored) sources (Dekker et al. 2018). There is therefore a strong possibility that this information is a) incomplete, b) unreliable and c) self-reinforcing. This could lead to “self-organized” migration routes that are sub-optimal for the asylum seekers on one hand and hard to predict and/or control for authorities on the other hand.

To test this hypotheses, an agent-based model of a population of migrants needs to be developed, in which agents base their decisions on their knowledge of the world. At a high level of abstraction, this knowledge is acquired by exploring the environment and by communicating between agents.

Such modeling effort typically involves translating a non-formal description of a system into concrete program code. Due to the ambiguity of the description, this might results in different implementations.

In addition, the medium of implementation, that is, the specific formalism into which a model description is translated can have a substantial influence on the modeling decisions. Therefore, two independent implementations of the same model description can differ considerably and might or might not give the same results. Since we also know that finding implementation errors in simulation code is notoriously hard, a good case can be made for independent implementations of the same model description.

Therefore, instead of focusing on a single model, we followed the pairs-of-models approach suggested recently (Zeigler 2017b). However, instead of aiming at multi-resolution modeling, and formally relating the models' properties by morphisms (Zeigler 2017a), here the pairs-of-models approach is aimed at avoiding too early commitments in our development process of the simulation model and elucidating central properties of the targeted simulation model. To capture the causal mechanisms behind such complex social processes as migration, an agent- or individual-base modeling approach is necessary (Willekens et al. 2018). Hence, we develop two agent-based models, rather than following a non-agent (e.g., systems dynamics) and an agent-based model (Morecroft and Robinson 2014). One will be implemented in a general purpose language (GPL) and will be on a discrete step-wise systems semantics. The other will be implemented in an external domain specific language (DSL) for agent-based models with continuous-time Markov chain (CTMC) semantics. Here, "external" means that the DSL defined independently of any other language, as opposed to an internal DSL, which makes use of the syntax of a host language into which it is embedded (van Deursen et al. 2000).

The code for both implementations is publicly available at https://github.com/mhinsch/RRGraphs_mini and https://github.com/oreindt/routes-rumours-ml3, respectively.

2 MODEL DESCRIPTION

The model consists of a population of agents migrating through a network of cities and transport links towards a target. Agents start out with no or very little knowledge about the world but can acquire knowledge either from their local environment or by communicating with other agents. Based on the information they have collected, they attempt to find the best path to one of the targets. In the following sections, we give a high-level overview of the model. Specifics of the implementations and their differences are discussed separately.

2.1 Environment

The simulated world consists of a random geometric graph (Gilbert 1961) of cities connected by transport links (see Figure 1). In addition there is a small number of entry and exit locations, respectively, representing border crossings. Entries and exits are connected by "in-official" and thus slow transport links to the nearest cities. Fast and slow transport links differ in their specific friction, which influences the resource costs associated with a link.

Cities have an inherent quality that represents how easy it is for refugees to stay in that city. This could include for example frequency of police controls, general safety, or availability of cheap accommodation. In addition, each city has an abstract resource availability (representing how easy it is to acquire food, money, clothes, etc.).

2.2 Agents

Agents at all times are either located in a city or transiting between cities. They enter the world by appearing at one of the entry points. They leave the world as soon as they arrive at an exit point, representing the destination countries. Agents have and collect information about the world. For each city and transport link they have either no information or estimates of the corresponding entity's properties together with trust values that indicate the assumed quality of each estimate. Agents keep in contact with a number of other agents, enabling them to exchange information. An agent's contacts can be active or inactive and can be

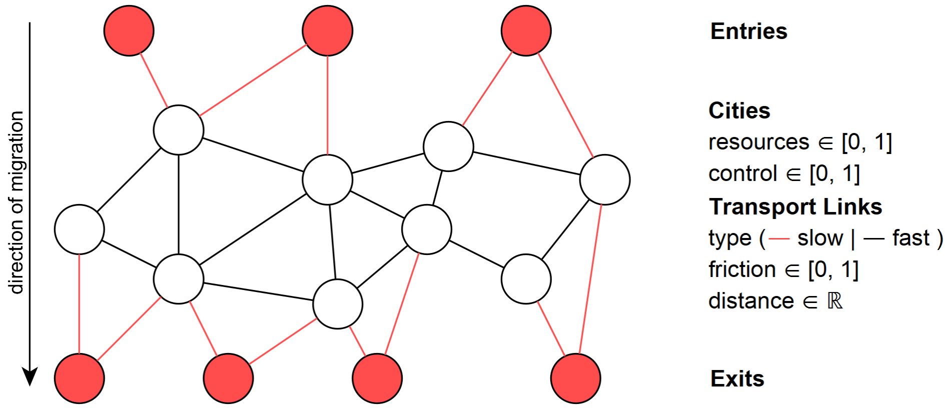

Figure 1: Sketch of the model environment.

located anywhere (i.e. not necessarily at the same location as the agent). During their journey agents use and replenish resources (money, food, etc.).

2.3 Agent Behavior

Agents explore their surroundings, move between locations and exchange information with their contacts. Ultimately, they attempt to find their way to one of the exit locations. They do that by comparing the desirability of their location and all adjacent cities (based on their current knowledge). The desirability of a city is a function of its specific quality, resource availability, proximity to the target and time required to reach it, all of it discounted by the agent's trust in the respective information. Then they preferentially move to the most desirable location (which might mean staying where they are).

Agents can exchange some of their information with their contacts. Information on locations or links that only one of two interacting agent is aware of is transferred directly. If both agents have information on a given feature they adapt their knowledge based on their respective beliefs and confidence. While staying in a city or traveling along a transport link, agents improve their knowledge about the respective location or link by increasing the accuracy of their estimates of the its properties as well as the corresponding trust values and by "discovering" geographically connected locations and links.

While staying in a city agents can increase their level of resources dependent on a city's resource availability. Traveling on a transport link on the other hand uses resources according to the link's friction.

3 IMPLEMENTATION

We have implemented the model in the general purpose language Julia (Bezanson et al. 2017) as well as in the Modeling Language for Linked Lives (ML3), a domain-specific language for agent-based modeling (Warnke et al. 2017).

Julia is a relatively new language designed specifically with applications in simulation and numerical computing in mind. Semantically it is close to the LISP language family, while at the same time attempting to be familiar to users of languages such as Matlab or Python in terms of syntax. Due to type inference and just-in-time compilation it can in many cases be as or nearly as efficient as C or C++. The language generally favors imperative over functional programming but provides very little support for object-orientation. It features an expressive type system, multiple dispatch, generic programming, and a powerful macro system. Of special interest for model development are Julia's optional typing and its REPL (read-eval-print loop) that enable rapid prototyping.

```
1  # model logic:
2  function step_agent_move!(agent, world, par)
3        agent.in_transit = true
4
5        loc = next_step(agent, world, par)
6        link = find_link(agent.loc, loc)
7        link.count += 1 # update traffic counter
8
9        costs_move!(agent, link, par)
10       explore_move!(agent, world, loc, par)
11       move!(world, agent, loc)
12 end
13
14 # corresponding scheduling logic:
15 function step_agent!(agent, model, par)
16       if decide_stay(agent, par) # model logic
17             step_agent_stay!(agent, model.world, par)
18       else
19             step_agent_move!(agent, model.world, par)
20       end
21       step_agent_info!(agent, model, par)
22
23       agent.steps += 1 # update the agent’s step counter
24 end
```

Figure 2: An excerpt of the Julia implementation of the model of migration routes showing the migrant's decision to move to another location. The function `step_agent_move` contains the consequences of that decision. The migrant will now be in transit, decides for a destination, has some costs deducted, explores, and is then moved. The decision to move happens (`decide_stay`) at a different place, in the function `step_agent` that determines the events happening to the agents, and their order (scheduling of events). In lines 8 and 24 the events are logged to observe the traffic on the link and the duration of the agent's travel.

ML3 is an external domain-specific modeling language built for agent-based demographic models. It is designed around the central concept of linked agents and their behavior. Agents' actions are described either as stochastic rules with CTMC semantics or as deterministically scheduled events. Common features of demographic models, such as age-dependent behavior and dynamic social networks, are directly supported by the language. ML3 models are interpreted by a Java-based simulator (Reinhardt and Uhrmacher 2017) using a variation of Gillespie's Stochastic Simulation Algorithm (Gillespie 1976). The language has been successfully applied to models of human migration and decision processes (Warnke et al. 2017; Reinhardt et al. 2018). However, the collection and adaptation of knowledge, a central part of this model, was not part of previous models.

3.1 Simulation Model 1: Julia

The Julia version of the model (see Figure 2 for an excerpt) handles all model processes (movement, exploration, information exchange) in discrete time steps. For events that do not happen in every time step (such as information exchange), fixed probabilities per time step were assumed. Building the model around events in continuous time would have been possible, but would have required more implementation effort.

Agents, cities, links and items of information are implemented as simple records (`mutable struct` in Julia) that are stored in arrays as part of a global model record. Each agent stores an item of information for each city and link in the world in an array, using the same order as the global storage. While this implementation scales very badly with number of cities and/or links, it is trivial to implement and does not present a performance issue for the type of scenarios we were interested in. In addition it allows for fast constant-time access to information items given a city's or link's global index. Each agent also maintains a list of references of agents it keeps in contact and is able to exchange information with.

```
1 Migrant
2 | ego.capital >= 0 && !ego.in_transit
3 @ ego.move_rate()
4 -> ego.in_transit := true
5    ego.destination := ego.decide_destination()
6    ego.capital -= ego.move_cost(ego.destination)
```

Figure 3: An excerpt of the ML3 implementation of the model showing the rule for deciding to move to another location. The guard (line 2) determines that only migrants with non-negative capital who are not already in transit to another location might decide to move. The rate of the rule (line 3) is calculated by a function, that takes the subjective quality of the current location and potential destinations into account. Lines 4–6 show the effect of the rule. The migrant is now in transit, decides for a destination, and has some cost deducted (compare with `step_agent_move` in Figure 2). The exploration during the move is implemented in a separate rule that might fire when the migrant is in transit.

The simulation engine iterates through all agents in each time step and executes their behavior. Actions that are performed at a rate lower than 1 are executed with a specific probability. Similarly, if two agents exchange information, we iterate through all items of information (i.e. a full list of all cities and links) and determine for each if one of the agents has knowledge about that part of the world and if so, whether an exchange happens.

To observe the model behavior, data is gathered during and at the end of the simulation. Most of the data is recorded by iterating through all agents/cities/links after each time step and once after the model run is finished. For some measurements (e.g. number of steps an agent needs to finish its itinerary, total number of visitors a city/link has during the runtime of the model), however, keeping track of them independently would have duplicated substantial parts of the model's data structures. In these cases we therefore added variables directly to the entity records and updated them during model execution (see Figure 2 line 24). The parameters of the simulation model are stored in an immutable record that is threaded as an argument through all function calls in the model code that need access to parameter values (`par` in Figure 2).

3.2 Simulation Model 2: ML3

As an agent-based formalism, the basic entities of ML3 models are agents. Hence, we implemented the migrants and cities as such. For relations between agents, ML3 uses the concept of links: bidirectional connections between agents, where the directions are distinguished by the role of the agents in the connection. These could be used to model the social network between the migrants, and for linking migrants to the city where they are currently located. However, the transport network between cities could not be realized using these features, as transport links are attributed, which ML3s links do not support. As a workaround, transport links are represented by "transport-agents", which carry the attributes, and are linked to the two endpoints of the transport link. The representation as agents would allow to equip transport links with a behavior of their own. However, in this version of the model this is not required.

ML3 does not distinguish between state and beliefs of an agent. Similar to the Julia model, we store a migrant's knowledge as a collection of information items. However, as ML3 does not support structured types or data structures apart from sets (which form the basis for ML3s links), these information items must be represented as separate agents, albeit without any behavior of their own. These "information-agents" contain the believed values and the agent's trust in that values as attributes, and are linked to the subject city or transport agent and to the migrating agent the beliefs belong to. The knowledge of a migrating agent is the set of information-agents it is linked to.

To model the agents' behavior, ML3 uses stochastic rules. Such a rule is a guard-rate-effect triplet, that is associated with a certain type of agents (see Figure 3). The guard is a condition that determines which agents of the given type have this behavior. The rate determines when the rule is fired, by giving the parameter of an exponential waiting time distribution. Finally, the effect determines what happens, when

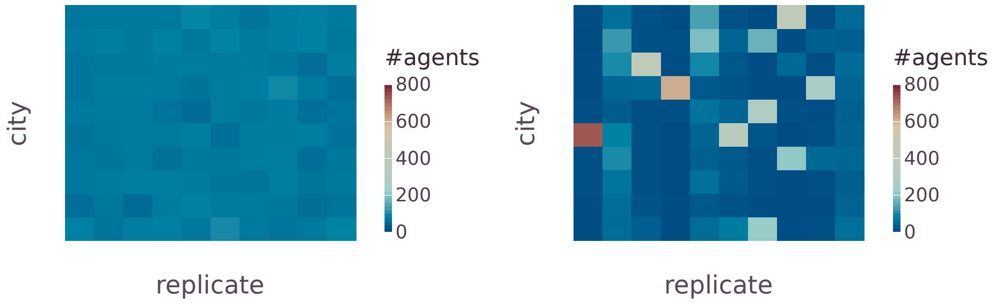

Figure 4: Number of agents passing through each exit city over the course of 100 time units for ten replicate runs (Julia model on the left, ML3 on the right).

the rule fires. In the example, migrants who have any capital left and are currently not in transit between two cities may decide to move to a neighboring city. The rate is determined by the function `move_rate`. In the effect, the migrant changes its state to be in transit, decides for a destination, and has the cost of that move deducted from their capital.

All other behavioral options are modeled similarly, resulting in a set of stochastic rules. The agents' decision between these options is then handled stochastically via a "competing risk" approach (see Klabunde et al. (2017)). Prioritization between the options is realized using the rates, so that more attractive options are given a higher rate. These options are therefore more likely to have the smallest waiting time, and are more likely to fire. However, this approach to decision making is rather limited, as the details of decision-making can not be modeled directly. In particular, the timing of decisions and the probabilities of deciding for the different options can not be decoupled, as the stochastic rates describe both timing (via the waiting times) and probability (as the probability of a certain rule firing first is determined by its rate in relation to the rates of all other rules).

3.3 Selected Results

To compare the behaviour and the runtime characteristics (see below) of the two implementations we ran the same scenarios over several replications in both simulation models. We found that the output differed significantly (see Figure 4). While in the ML3 model most of the migrants arrive at a few exits, with the specific exits varying between replications, in the Julia model, the migrants are spread more evenly between exits.

4 EVALUATION

We compare the two model implementations according to a few selected criteria. We start each paragraph arguing why we consider this aspect relevant before assessing the two implementations.

4.1 Separation of Model and Simulator

Strictly separating model and simulation logic provides significant advantages to many aspects of a simulation study (Zeigler et al. 2000). A clear separation between model and simulator makes the model implementation more succinct, and therefore more accessible and maintainability. As there is less implementation effort for the modeler, there is less room for errors. Additionally, as the simulation logic is now independent of the model, it can be reused for multiple models. This allows for putting more effort into implementing and maintaining the simulation algorithms, improving software quality and enabling the implementation of advanced simulation algorithms.

Due to its more ad-hoc nature, in the Julia implementation, model and execution are tightly interwoven (see Figure 2). In ML3 they are strictly separated by design. The modeler only describes the model behavior declaratively in terms of stochastic rules. The model is executed by the ML3 simulator, which implements the actual scheduling and execution of events.

4.2 Developing Models by Composition and Extension

Building and validating more complex models is usually done iteratively (Cioffi-Revilla 2010), by successively composing or extending simpler models. Easy extension or composition of models should therefore be supported by the implementation language.

In ML3, although no black box composition, as e.g. provided by the DEVS formalism (Zeigler et al. 2000), is supported, models can easily be composed or extended in a white box manner by adding behavioral rules. The extension of models realized in Julia requires more care as the changes to be done are not as localized as in ML3: A consequence of model and simulation code being closely intertwined. Furthermore Julia's type system makes it difficult to separate model components into independent modules - required to build models by composing them from simpler models - without resorting to expensive runtime polymorphism.

If the models are composed (irrespective of the manner of composition) typically each model component is validated independently. Simulation experiments can then be reused to validate the composed model (Peng et al. 2017). In our case we might, for example, develop a component model concerning the departure of migrants from their home country and validate the produced rate of departures over time. After composing it with the routes model - replacing the simple arrival process introducing new migrants to entries - the rate of departures should remain the same and could be re-validated with the same experiment. If successively extending a model, old simulation experiments can therefore be reused in a manner similar to regression testing to test whether the newly extended model still fulfils the validation experiments conducted previously. For this a clear separation of concerns is beneficial as well.

4.3 Flexibility for Adding Model Features

An iterative extension-based model requires a considerable amount of flexibility and expressiveness of the chosen simulation environment. It must allow for including model features that were initially not planned. The latter is usually not an issue when using a general purpose language. Domain-specific modeling languages, especially external ones, however, are by design limited in their expressiveness.

It is, of course, inherently difficult to make predictions about future model extensions. One extension we have in mind is to have migrants make plans about their future path, which might affect how stable routes form. While we can add this to the Julia implementation, the lack of structured types and data structures might make that impossible or at least cumbersome in ML3, similarly as the knowledge representation as agents in our example.

4.4 Language Infrastructure

Infrastructure surrounding a language plays an important role for the language's usability. Established general purpose languages typically come with a variety of software tools like IDEs and debuggers to support development, as well as ample documentation, libraries that solve common problems, and a large community of users to exchange experience.

As an established, albeit relatively young, language, Julia and its ecosystem offer many of these. The possibility to execute parts of the Julia code via the command line proved invaluable for testing purposes. Julias built-in documentation system helps creating an accessible documentation of the implementation. Jupyter (Ragan-Kelley et al. 2014) enables literate programming with Julia.

In terms of tool support, ML3 only offers an editor with syntax highlighting. Documentation about the language is available in form of publications and existing models.

4.5 Simulation Infrastructure

A simulation model has little use without the means to perform simulation experiments with it. Such experimentation consists of a wide range of different interdependent tasks, e.g., parameterization of the model, making observations, and storing and analyzing results. In addition, for the purpose of reproducibility, simulation experiments must be comprehensible and repeatable by different people on a different machine. To support a systematic successive refinement intertwined by phases of validation, it is also essential to separate the different aspects of simulation software, i.e., model, execution and simulation experiments, to enable a reuse of simulation experiments (Peng et al. 2017). Consequently, software solutions are necessary.

As a general purpose language Julia offers no specific support for simulation infrastructure. The language proved versatile enough, however, that ad-hoc solutions could easily and quickly be implemented. We simply let the source file that contains the definition of the parameters record (see above) double as a configuration file for the simulation. Due to Julia's just-in-time compilation model this does not reduce (run time) performance. Furthermore the `Parameters.jl` package provides a compact syntax to add default values to record fields so that the syntactical overhead compared to a dedicated configuration file is minimal. We further used Julia's meta-programming and reflection capabilities (in combination with the `ArgParse.jl` package) to optionally read the value of any parameter from the command line (again with no performance costs). This allowed us to run simulations on an HPC cluster using a simple, pre-existing Ruby package by one of the authors (MH) that reads parameter combinations from a description file and generates the necessary shell scripts for a cluster environment.

Due to the language's focus on scientific computing and the existing infrastructure for interactive development it was furthermore straightforward to perform the analysis of the model results including the generation of figures for this publication in Julia as well (using the Jupyter notebook interface). As the language is intended for the use in scientific computing, the Julia ecosystem can be conveniently exploited for conducting simulation studies.

In ML3, simulation experimentation is realized via a binding to SESSL (Ewald and Uhrmacher 2014), a domain specific language for specification of simulation experiments embedded into the GPL Scala. SESSL, allows to specify simulation experiments in a declarative and executive manner, including, e.g., parallel execution of replications, parameterization and experimental design, some methods for statistical analysis, and export of simulation results for analysis with external tools (Reinhardt et al. 2018). Experiments specified in SESSL are directly executable Scala code. Observation of model behavior can be specified declaratively in SESSL as well. Therefore, observation logic, e.g., counting the number of arrivals, is not mixed with the model logic itself (compare Figures 2 and 3).

4.6 Runtime

Execution time is a significant limiting factor for many simulation efforts, in particular with agent-based models (Collier and North 2013). Agents' behavior is complex, and to see effects emerging from interactions, a sufficiently large population of agents and a sufficiently large environment are necessary. As complex models tend to have many parameters the number of combinations that must be tested can be substantial. In addition to that, agent-based models usually involve stochastic processes, so that replications will be needed for each of these parameter combinations. For the calculation of replications, parallel computing infrastructures can be easily exploited. The number of parameter configurations to be calculated can be significantly reduced by relying on appropriate experimental designs and statistical techniques (Kleijnen 2015). However, the issue of executing simulations efficiently will always remain relevant.

We compared runtime of the two model implementations by running both with identical parameter values on the same machine (Intel I7 8550U, Ubuntu 18.4). For a small world with only 50 cities and with 10 new agents entering the world per time unit, running the model for 100 time units took about 10s for the Julia version (8s of which were spent JIT compiling the code) and 4450s for the ML3 version.

Some of this difference can be attributed to the execution model of the respective languages. ML3 code is directly interpreted by the simulator while Julia code is compiled to machine binary before execution. It has been shown that even the compilation of only critical pieces of a model at runtime can yield significant efficiency gains (Meyer et al. 2018).

However, the main problem lies in the realization of the knowledge gathering and analysis in the two modeling approaches. A majority of the simulation time is spent on the information exchange between migrants. In Julia we were able to chose appropriate data structures to solve this efficiently (albeit at the cost of greater implementation effort). In ML3, however, the modeler lacks this level of control. Here, all collections of agents (or in this case pieces of information) are simple sets. Consequently, the implementation scales very badly with the amount of knowledge agents have. E.g., specific items of information can only be selected in linear time by iterating over these sets.

4.7 Simulation results

The simulation results produced by the two implementations differ significantly. It raises concerns that the origin of these differences it is not immediately evident. E.g., they might be a result of the different underlying execution semantics or the result of divergences in the implementation of the model logic itself.

Closer analysis and comparison of the formal models implemented by the two simulation models is required to gain further insight. The formal semantics of the ML3 implementation is quite clear, as the language is based on a CTMC semantics, or more generally, if all language features were used, a Generalized Semi-Markov Process (Glynn 1989). The Julia version on the other hand, as most ad hoc model implementations, lacks a formally defined semantics. It can formally be seen as a discrete-time Markov chain (DTMC), however, the precise DTMC described is not immediately obvious. While in principle approximating one with the other is possible (Doytchinov and Irby 2010), the lack of explicit formalization on the Julia side makes a comparison very difficult. As an alternative, the semantics of the implementations can be brought closer together by lowering the step-size and transition probabilities of the Julia implementation, matching the transition rates of the ML3 implementation. This should lead to convergence of the behavior of both versions if they indeed implement the same abstract model.

In addition to these static analyses, additional simulation experiments are needed to elucidate behavioral properties and possible idiosyncrasies of the two models. It should be tested whether and how fast in both models optimal paths within the environment will be found by perfect communication, or perfect exploration; whether in an environment where little differences in routes exist, a uniform distribution of migration routes emerges; and how sensitive the models react to changes in communication and exploration.

5 DISCUSSION

We have implemented an agent-based model of migration route formation twice: Once in the GPL Julia and once in the external DSL ML3.

In direct comparison, there are clear trade-offs between both approaches, each of which has their own advantages and limitations. The ML3 implementation demonstrates that especially the strict separation of different simulation concerns – model, execution, experimentation – is advantageous. Having model and simulator separated not only eases the implementation of the former, the increased accessibility and maintainability facilitate further development of the model. Furthermore, we can easily take advantage of existing simulation infrastructure surrounding the language. Having a language with a strict paradigm (here the continuous-time stochastic rule approach) that must be followed does aid – or force – the modeler to formalize the model precisely.

However, these advantages over implementing the model from scratch come at a cost. Firstly, we have found a significant runtime cost. It should be noted this is not a consequence of using a DSL instead of a GPL. However, as soon as a feature that is not directly supported in the DSL is expressed by other means, this will come at significant performance cost. Secondly, we have no access to the extensive software ecosystem surrounding an established GPL, which is a significant practical disadvantage. Finally, using a GPL results in greater flexibility, especially for extending the model with new features. All considered, neither of the approaches is ideal. While the advantages in methodology of applying a DSL are considerable, the limitations of the language at hand make its application impractical for a model that relies heavily on complex data structures. Human decision-making processes are complex, and a sufficiently expressive language is necessary to model them.

An extension of ML3 for the inclusion of additional features for data structures would result in the necessary expressiveness. This additional control would also allow the modeler to chose an implementation that is more runtime efficient. Although there exist methods that support the evolution of domain-specific languages (Thanhofer-Pilisch et al. 2017), the adaptation of an external DSL to new requirements implies larger efforts. A language workbench (Erdweg et al. 2013), a software framework for developing DSLs and tool support for DSLs, could be employed to successively evolve the language. This would require a re-implementation of parts of ML3, but techniques for migrating custom DSL implementations to a language workbench exist (Denkers et al. 2018). Most of the simulator implementation might even be reused as is, as it is mostly independent of the surrounding infrastructure for, e.g. parsing and type checking. However, this extension of ML3 would mostly consist of adding features typically found in GPLs.

Therefore, an alternative approach would be to continue the development as an internal DSL, i.e., to embed the DSL into a host language (van Deursen et al. 2000). Thereby the DSL, and by extension the user, has access to host language features. In addition, host language tools, e.g., compiler, debugger, or IDEs, can be reused. It has been demonstrated (Warnke et al. 2016) by embedding a simplified rule-based language for agent-based models into the GPL Scala, that it is feasible to have a compact declarative description, while having access to GPL features. For ML3 this might be especially suitable, as the syntax and semantics of ML3 is already close to many object-oriented languages, making the embedding potentially very natural.

Either way, integrating support for decision making and learning might be desirable, as this part of the model is conceptually challenging, and difficult to implement efficiently. The representation of knowledge we chose was as much driven by the practicalities of model implementation as by the requirements of the model. Replacing the ad-hoc solution with a formal model also has an epistemological advantage. A wide variety of formal approaches for modeling decision making exist (Balke and Gilbert 2014), that could form the foundation for including decision making into the language.

Going beyond the specifics of the two versions of the model, implementing it in parallel in two different languages based on two different paradigms has proven a valuable exercise. First, comparing both implementations gives a clear indication of the advantages and disadvantages of either approach. Furthermore it becomes obvious that different paradigms lead to different modeling decisions. Finally, whether the differences in results can be traced back to the different paradigms of the two implementations or to subtle differences in the implemented abstract models - in either case assumptions that clearly affect the model behavior and that would have remained implicit for a single implementation will have become explicit. Thereby, simulation results are put into question, motivating additional efforts to explore and analyse the behavior of the respective models to explain these differences.

ACKNOWLEDGMENTS

This research was supported by the European Research Council (ERC) via research grant Bayesian Agent-based Population Studies (CoG-2016-725232).

REFERENCES

Balke, T., and N. Gilbert. 2014. “How Do Agents Make Decisions? A Survey”. *Journal of Artificial Societies and Social Simulation* 17(4):13.

Bezanson, J., A. Edelman, S. Karpinski, and V. Shah. 2017. “Julia: A Fresh Approach to Numerical Computing”. *SIAM Review* 59(1):65–98.

Borkert, M., K. E. Fisher, and E. Yafi. 2018. “The Best, the Worst, and the Hardest to Find: How People, Mobiles, and Social Media Connect Migrants In(to) Europe”. *Social Media and Society* 4(1).

Cioffi-Revilla, Claudio 2010, January. “A Methodology for Complex Social Simulations”. http://jasss.soc.surrey.ac.uk/13/1/7.html.bak.

Collier, N., and M. North. 2013, October. “Parallel Agent-Based Simulation with Repast for High Performance Computing”. *SIMULATION* 89(10):1215–1235.

Dekker, R., G. Engbersen, J. Klaver, and H. Vonk. 2018. “Smart Refugees: How Syrian Asylum Migrants Use Social Media Information in Migration Decision-Making”. *Social Media and Society* 4(1).

Denkers, J., L. van Gool, and E. Visser. 2018. “Migrating Custom DSL Implementations to a Language Workbench (Tool Demo)”. In *Proceedings of the 11th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2018, 205–209. New York, NY, USA: ACM.

Doytchinov, B., and R. Irby. 2010. “Time Discretization of Markov Chains”. *Pi Mu Epsilon Journal* 13(2):69–82.

Ekman, M. 2018. “Anti-refugee Mobilization in Social Media: The Case of Soldiers of Odin”. *Social Media and Society* 4(1).

Erdweg, S., T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. D. P. Konat, P. J. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. A. Vergu, E. Visser, K. van der Vlist, G. H. Wachsmuth, and J. van der Woning. 2013. “The State of the Art in Language Workbenches”. In *Software Language Engineering*, edited by M. Erwig et al., Lecture Notes in Computer Science, 197–217: Springer International Publishing.

Ewald, R., and A. M. Uhrmacher. 2014. “SESSL: A Domain-Specific Language for Simulation Experiments”. *ACM Transactions on Modeling and Computer Simulation* 24(2):11:1–11:25.

Gilbert, E. 1961. “Random Plane Networks”. *Journal of the Society for Industrial and Applied Mathematics* 9(4):533–543.

Gillespie, D. T. 1976, December. “A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions”. *Journal of Computational Physics* 22(4):403–434.

Glynn, P. 1989, January. “A GSMP Formalism for Discrete Event Systems”. *Proceedings of the IEEE* 77(1):14–23.

Klabunde, A., S. Zinn, F. Willekens, and M. Leuchter. 2017, October. “Multistate Modelling Extended by Behavioural Rules: An Application to Migration”. *Population Studies* 71(sup1):51–67.

Kleijnen, J. P. C. 2015. *Design and Analysis of Simulation Experiments*. 2 ed. International Series in Operations Research & Management Science. Springer International Publishing.

Leurs, K., and K. Smets. 2018. “Five Questions for Digital Migration Studies: Learning From Digital Connectivity and Forced Migration In(to) Europe”. *Social Media + Society* 4(1):205630511876442.

Massey, D. S., J. Arango, G. Hugo, A. Kouaouci, A. Pellegrino, and J. E. Taylor. 1993. “Theories of International Migration: A Review and Appraisal”. *Population and Development Review* 19(3):431.

Meyer, T., T. Helms, T. Warnke, and A. M. Uhrmacher. 2018. “Runtime Code Generation for Interpreted Domain-Specific Modeling Languages”. In *Winter Simulation Conference (WSC 2018)*, 605–616. Göteborg, Sweden: IEEE.

Morecroft, J., and S. Robinson. 2014. “Explaining Puzzling Dynamics: A Comparison of System Dynamics and Discrete-Event Simulation”. In *Discrete-Event Simulation and System Dynamics for Management Decision Making*, 165–198. John Wiley & Sons, Ltd.

Peng, D., T. Warnke, F. Haack, and A. M. Uhrmacher. 2017. “Reusing Simulation Experiment Specifications in Developing Models by Successive Composition — a Case Study of the Wnt/β-Catenin Signaling Pathway”. *SIMULATION* 93(8):659–677.

Ragan-Kelley, M., F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonnier. 2014. “The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication.”. In *AGU Fall Meeting Abstracts*.

Reinhardt, O., J. D. Hilton, T. Warnke, J. Bijak, and A. M. Uhrmacher. 2018. “Streamlining Simulation Experiments with Agent-Based Models in Demography”. *Journal of Artificial Societies and Social Simulation* 21(3):9.

Reinhardt, O., and A. M. Uhrmacher. 2017. “An Efficient Simulation Algorithm for Continuous-Time Agent-Based Linked Lives Models”. In *Proceedings of the 50th Annual Simulation Symposium*, ANSS ’17, 9:1–9:12. San Diego, CA, USA: Society for Computer Simulation International.

Thanhofer-Pilisch, J., A. Lang, M. Vierhauser, and R. Rabiser. 2017, August. “A Systematic Mapping Study on DSL Evolution”. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 149–156.

van Deursen, A., P. Klint, and J. Visser. 2000, June. “Domain-Specific Languages: An Annotated Bibliography”. *SIGPLAN Notices* 35(6):26–36.

Warnke, T., O. Reinhardt, A. Klabunde, F. Willekens, and A. M. Uhrmacher. 2017, October. “Modelling and Simulating Decision Processes of Linked Lives: An Approach Based on Concurrent Processes and Stochastic Race”. *Population Studies* 71(sup1):69–83.

Warnke, T., O. Reinhardt, and A. M. Uhrmacher. 2016. “Population-Based CTMCs and Agent-Based Models”. In *Proceedings of the 2016 Winter Simulation Conference*, WSC ’16, 1253–1264: IEEE Press.

Willekens, F. et al. 2018. “Towards causal forecasting of international migration”. *Vienna Yearbook of Population Research* 16:1–20.

Zeigler, B. P. 2017a. “Constructing and evaluating multi-resolution model pairs: an attrition modeling example”. *The Journal of Defense Modeling and Simulation* 14(4):427–437.

Zeigler, B. P. 2017b. “Why should we develop simulation models in pairs?”. In *2017 Winter Simulation Conference, WSC 2017, Las Vegas, NV, USA, December 3-6, 2017*, 2: IEEE.

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

AUTHOR BIOGRAPHIES

OLIVER REINHARDT is a Ph.D. student in the modeling and simulation group at the University of Rostock. He holds an MSc in Computer Science from the University of Rostock. In his research, he is concerned with domain-specific modeling languages and the methodology of agent-based simulation. His email address is oliver.reinhardt@uni-rostock.de.

MARTIN HINSCH is a research fellow in the Department of Social Statistics and Demography at the University of Southampton. He is interested in emergent structures and complex systems and has done research in theoretical biology, bioinformatics, machine learning, epidemiology and swarm robotics. His email address is hinsch.martin@gmail.com.

JAKUB BIJAK is Professor of Social Demography and joint Head of Department of Social Statistics and Demography at the University of Southampton. Currently, he leads an ERC funded project on Bayesian Agent-Based Population Studies. His email address is j.bijak@soton.ac.uk.

ADELINDE M. UHRMACHER is professor at the Institute of Computer Science of the University of Rostock and head of the Modeling and Simulation Group. She holds a PhD in Computer Science from the University of Koblenz and a Habilitation in Computer Science from the University of Ulm. Her email address is adelinde.uhrmacher@uni-rostock.de.