Reproducible Simulation Experiments in Agent-Based Demography

ABSTRACT

The number of agent-based models is steadily increasing. Not only in areas as demography, in which traditional data-driven, statistical approaches prevail, the hypothesis-driven design of agent-based models leads to questioning validity of these models. Consequently, suitable means to increase the confidence into models and simulation results are required. Here explicit, reproducible simulation experiments play a central role. However, more complex models often require diverse experimentation methods, and thus a flexible simulation environment. With a binding between SESSL – an internal domain-specific language for simulation experiments – and ML3 – a simulator for linked lives in demography – we provide a powerful simulation tool that can serve as a foundation for current efforts of employing advanced and statistical model analysis of agent-based demographic models. We demonstrate the benefit in specifying and executing different experiments with a health care model, and in documenting and reproducing simulation experiments.

CCS Concepts

•**General and reference** → **Experimentation;** Design; •**Computing methodologies** → **Modeling methodologies;** *Model verification and validation; Agent / discrete models;* •**Software and its engineering** → **Domain specific languages;**

Keywords

reproducibility; demography; agent-based modeling; experimentation

1. INTRODUCTION

The use of agent-based computational models in demography dates back to the seminal 2003 book edited by Francesco Billari and Alexia Prskawetz [6]. This research area has gained momentum especially over the past few years, with some of the most recent advances in the field reported in [34]. Still, there remain tensions between the micro-level explanations of demographic phenomena and the macro-level population patterns that are of interest to most demographers (see [7] for an excellent overview). One way of reconciling the tensions between these two levels is to carry out computer experiments, driven by the principles of statistical design of experiments, in order to systematically study the macro-level outcomes of micro-level processes. A lot of work in this area has focused on the statistical aspects of the analysis of experiments, including the use of polynomial [22] or interval [17] regression meta-models for this purpose, and the use of Minimum Simulated Distance approaches for calibration [12]. The techniques of [20] based on Gaussian Processes have been applied to demographic questions, for example in [4] and [16].

The existing software for simulation experiment design and execution typically either require specialist knowledge and programming skills, or have very limited functionality. Hence, the aim of this paper is to present a *flexible* and *efficient* environment for supporting computational experiments, which also aids the reproducibility of the results. Here, we refer to flexibility in terms of being able to conduct a broad range of simulation experiments with diverse characteristics, and to efficiency in terms of clear and concise communication of experiment specifications.

The existing practice with respect to experimenting with and validating agent-based models in demography is varied, and summarized in Table 1. The table categorizes a selection of existing agent-based demographic models according to a rough typography, describing various approaches to experimentation, calibration, validation, and documentation. The survey is not comprehensive, and exact classification of any given model on all dimensions is not possible, so the categories given are somewhat fuzzy. More detailed descriptions of the approaches taken are given in the following paragraphs.

The relative youth of the approach within the discipline and the fact that it is not yet an accepted mainstream tool means that no one single approach to these tasks exist, and, given that the suitability of methods depends on the model and the research questions to be asked, is likely to remain this way. Early, pathbreaking approaches were understandably rather ad-hoc - for instance, in [5], a model of marriage and social pressure, results are reported at a set of default parameters, and are also provided for a small number of additional scenarios. Similarly, the simulation in [15] models partnership formation as a search and match process where agents aim to find agents similar to themselves, but relaxed their criteria as to what constitute a good match over time.

PADS '17 May 24–26, 2017, Singapore

© 2017 ACM. ISBN 123-4567-24-567/08/06. . . $15.00

DOI: 10.475/123_4

Stylized Type	**Experimentation**	**Calibration**	**Validation**	**Documentation**	**Examples**
Ad-hoc	Small number of scenarios tested. No systematic attempt to explore the spaces	Ad-hoc methods of finding suitable parameters	Qualitative match with observed data	Textual description of model parameters	[5] [15]
Systematic variation	One-parameter-at-a-time variation or grid-based design	Optimization over a grid of points	Quantitative match with observed data	Parameters provided and ODD description of model	[1] [11] [21] [4]
Model-based	Central Composite Design or Latin Hypercube Sample design	Model-based e.g. parametric or semi-parametric regression meta-models	Match to hold-back or unseen data	Code and executable experiment scripts provided (e.g. R or NetLogo BehaviorSpace)	[18] [14] [16]
Gold standard	Flexible / Optimal / Sequential designs	Model-based - fully accounting for uncertainty	Hold-back or unseen micro-data / different contexts	Succinct, integrated, and readable experimentation and execution code	

*Some articles are difficult to classify. For example, [21] provides all necessary code and scripts for the running experiments, but for experimentation and calibration criteria falls in the 'systematic variation' category. Likewise, [4] used grid-based design for experiments, but fitted Gaussian process emulators to analyze the model uncertainty. On the other hand, in the fully stochastic [16] there is no systematic validation attempt.

Table 1: Survey of Existing Practice in Demographic ABM

Their experiments aimed at examining the theoretical consequences of greater population heterogeneity and cultural diversity on the matching process by varying the relevant parameters one at a time. They are able to qualitatively match US marriage curves (although no detail as to how this was achieved is given), and they also recreated the divorce rate using the same parameter settings as an attempt at model validation.

Later examples are more systematic in their approaches to experimenting with simulations. Most often, the parameter set which minimizes some distance to observed quantities is sought. For instance the model in [1] examines the effect of social network pressure on transition to first birth in Austria, and replicate changes in the timing of first birth seen in Austria in the 2000s. A metric measuring the distance between simulated and real Age-Specific Fertility Rates is used to assess model performance. A larger set of experiments are attempted in this work, with combinations of parameters evaluated over a grid, and a 'null' model in which the network effects are turned off is also investigated. Presumably, the stated default parameters are those on the grid for which this distance is minimized. However, because of the practice of varying two parameters while keeping the others fixed, large areas of the parameter space were left unexplored.

Other work [11] advances this agenda by examining the effect of social networks on the success of family policies. The authors modeled preferred family size as dependent on opinions of social network members. This time, social network growth is endogenous and dependent on agent similarity, degree of relatedness, and number of shared network contacts. A grid search over 6 parameters leads to a total of 741,312 simulations. More recently, [18] examines sex-selective abortion within the analytical framework of 'ready, willing, and able' [9], showing the dynamic relationship between son preference, social pressure relating to family size, and diffusion of abortion technologies. The model is calibrated by fitting a regression meta-model to a Latin Hypercube sample of points, and then use these predictions to minimize the predicted root-mean-squared error of their model relative to observations.

Other papers have also examined the behavior of simulations over a grid of points, including [21], which examined circular migration between Mexico and the USA, modeling agents as deciding whether or not to migrate using a discrete choice framework. Following [39], some parameters of the model are fixed empirically using Mexican Migration Project data, whilst others were based on behavior rules, the parameters for which were found through a grid search of possible combinations, with a goodness of fit metric forming the criterion of choice. In contrast, and focusing on phenomena of assortative mating, [14] develops an agent-based model examining how increased educational attainment amongst women has led to changes in the marriage market. This simulation is calibrated against empirical data using regression metamodels fit on central composite designs, and validated against a hold-back set of data from other countries.

The above examples show that experimentation with agent-based models is important in addressing the sorts of questions typically of interest to demographers. However, doing so is often difficult and time-consuming, and often available tools only support simple combinatorial grid designs by default (e.g. [40]). Furthermore, agent-based models of demographic process are not always produced with reproducibility in mind, and simulation code and descriptions of the experiments conducted is often not provided or incomplete, with notable exceptions including [14, 21]. In part, this may be because of the difficulty in easily specifying and sharing sets of experiments of the nature required. Provision of a simple and flexible way of describing, sharing and running experiments with agent-based computational models of demographic processes would therefore support further development of the approach within demography.

The remainder of the paper is structured as follows. In section 2, the documentation needs for experimenting on agent-based models are outlined, followed by an introduction of an Scala layer for specifying and executing simulation experiments (SESSL - Simulation Experiment Specification via a Scala Layer). The test model, i.e., the linked lives model [27], which has been translated into ML3 [36] is described in section 3. Section 4 presents the SESSL binding to the ML3 simulator. As an illustration, experiments with the linked lives are made. The results are discussed in the concluding section 5.

2. DOCUMENTATION OF EXPERIMENTS

Supporting a variety of simulation experiments in a flexible manner is central for developing valid agent-based demographic models. Similarly important is an accurate and complete record of all the conditions that define a simulation experiment. Having such complete records of a (simulation) experiment increases its scientific rigor and, consequently, also its credibility, and is a pre-requisite for its reproducibility. The review about benefits and limitations of the ODD protocol (Overview, Design concepts, and Details), which was published in 2006 to standardize the description of agent-based models [13], recommends urgently to enhance ODD by a separate section on simulation experiments. Information about experiments done with a model facilitates assessing the range for which a model might be valid, interpreting and questioning published simulation results, and as such is essential in valuing and reusing models. Therefore, the basic information a simulation experiment description should contain needs to be identified. The *Minimum Information About a Simulation Experiment* (MIASE) standard proposed by [23] requires information about (1) the composition of the model that is simulated together with its configuration parameters, (2) the simulation methods used (incl. configuration, e.g., termination conditions), (3) the collection of tasks performed in the experiment, and (4) the complete collection of outputs that is produced. Similar information is required by the *Minimum Simulation Reporting Requirements* introduced in [29]. In addition, due to the focus on stochastic simulation, information about the pre-processing used to generate input data for the experiment, number of iterations of the experiment, information on random number generation and confidence levels used for estimation, and post-processing performed on the output data shall be included.

These standards set a high bar for reporting requirements and thus, suitable computational support is needed. The use of workflow management systems for modeling and simulation supports the user in creating well structured and documented experiments. With the use of workflows, the process of experimentation can be built into the tools by the system designer, as in [30] or [31], or offered to the simulationist as a series of templates to be completed, as in [32]. For specifying the workflows these systems rely on languages such as BPEL/BPMN [38]. Other domain specific languages directly aim at specifying and executing simulation experiments by direct bindings to or implementations within simulation and analysis tools, side stepping the use of specific workflow management systems. E.g., motivated by requirements in MIASE the SED-ML Simulation Experiment Description Markup Language has been developed for exchanging, encoding, and documenting [35] experiments, SED-ML is supported by a variety of simulation tools in systems biology. These domain specific languages appear particularly suitable for specifying simulation experiments that run in batch mode. One further representative of this class of domain specific language for simulation experiment specification and execution is SESSL [10] which is in contrast to SED-ML an internal domain specific language, and instead of the community effort SED-ML, which functions also as an exchange format, takes the form of a simulation system agnostic layer between user and simulation system.

2.1 Experimentation with SESSL

Due to the number of involved actors, community standards such as SED-ML typically evolve slower than the domain they aim at. The Simulation Experiment Specification on a Scala Layer (SESSL) [10] has been developed to mitigate this problem by making the experiment specification itself executable, allowing users to add code that is executed during the experiment. Thus, proficient users can directly add missing features "on-the-fly" instead of requesting their addition in a future version of the standard. As a domain-specific language embedded in the programming language Scala (www.scala-lang.org), SESSL offers many extension points where user-supplied functionality can be injected.

Although SESSL specifications are valid, executable Scala code, the resulting experiment specifications do not resemble typical program code. The salient aspects of simulation experiments, such as the model input parameter configuration or the observation of model outputs, are specified in a declarative style. But vanilla Scala functions, for example to post-process the observed output, can still seamlessly be integrated if needed.

Larger chunks of code can be packaged in a reusable binding. SESSL bindings are particularly useful to integrate third-party tools and make their features available in experiment specifications. Bindings to external tools are mostly translating elements of a SESSL specification to invocations of external tools. Thus, they are slim and easily implemented. When setting up an experiment, users can choose from the available bindings to enrich their experiment with features and connect it to one or more external tools. SESSL relies on Apache Maven (maven.apache.org) for the automatic resolution of dependencies and download of software artifacts for bindings and third-party software.

To structure the different methods that might be employed by concrete simulation experiments, SESSL makes use of Scala's traits. Traits can be "mixed in" when creating an experiment object, making the functionality of the trait available to the experiment. This makes traits the premier way to publish features of bindings. For example, a binding to a simulation package may provide traits for parallel execution, observation, or report generation; a binding for statistical model checking may provide a trait that allows specifying model behavior in a temporal logic formula.

Traits and bindings make the specification of simulation experiments in SESSL flexible and agile. Due to the automatic management of software artifacts, executing and repeating SESSL experiments is straightforward across machines and platforms. At the same time, SESSL specifications are succinct and readable, allowing easy sharing and commu-

category	careNeedLevel	weekly care hours
none	0	0
low	1	8
moderate	2	16
substantial	3	30
critical	4	80

Table 2: The five levels of care need, the according value of the care need attribute in ML3, and the amount of required care hours for the purpose of the calculation of the global care cost. [27]

nication of experiment specifications. This makes SESSL an excellent tool for experiment specification in domains that require diverse experiment design methods and non-standard approaches, such as agent-based computational demography.

3. THE LINKED LIVES MODEL IN ML3

To show our approach we will specify and execute some experiments on the linked lives social care model by Noble et al. [27], a discrete-time agent-based model of the UK population implemented in Python, that aims at capturing the effects of the aging population and changing family structures on the cost of state-funded social care. It focuses on the demographic processes that affect the supply and demand of social care. In the model the population of the UK is represented by agents with a scaling factor of $1 : 10,000$, i.e. one agent represents $10,000$ people in the real world. For all agents life-course transitions that are important for social care, e.g., fertility, mortality, partnership formation and internal migration, are simulated. Agents are born as dependent children of their parents. With age 17 they reach adulthood. At this point they enter the workforce and become a taxpayer. They may marry another agent, move to a different part of the UK, and get children of their own. When they reach the retirement age, they retire and leave the workforce. Anytime during their life their health status might degrade, which increases their need for social care. Finally, they will die, which removes them from the population. For the purpose of internal migration the model divides the UK into a grid of towns, with consist of multiple houses. Agents may migrate to a different house in the same town or to a different town multiple times in their life.

The central output variable of the model is the total cost of social care per taxpayer. Every agent needs a certain amount of social care per hour, depending on their care need category (table 2). A part of this care need can be fulfilled informally by relatives, the rest needs to be paid for by the state. Individuals who need no or only little care themself can provide informal care. However, they will not deliver care to anybody, but only to persons living in the same household and to their parents, as long as the parents live in the same town. Therefore, household structures and mobility affect the amount of informal social care that is actually delivered.

We reimplemented the model in the Modeling Language for Linked Lives (ML3), a domain specific modeling language for continuous-time agent-based demographic models, where agents interact in a social network. Every ML3 model consists

```
1  Person(sex:{"m","f"},
2    status:{"child", "adult at home", "independent
         adult", "retired"} := "child",
3    careNeedLevel:int := 0,
4    sec:int);
5
6  Couple();
7  couples:Couple[0-] <-> [2]Person:spouses;
8  parents:Couple[1] <-> [0-]Person:children;
9
10 House(sec:int);
11 Town(x:int, y:int);
12 house:House[1] <-> [0-]Person:occupants;
13 town:Town[1] <-> [1-]House:houses;
```

Figure 1: Declaration of agent types and links.

of three different components. Firstly, all acting entities of the model are represented as *agents*. Secondly, relationships between agents are described by bidirectional *links*. Finally, *rules* describe the agents' behavior. An in depth description of ML3 and a first reimplementation of the linked lives model is already given in [36]. However, as that implementation is based on a first concept of ML3 that was not yet executable, some aspects of it have changed. Here we will only show central ideas and the matchmaking for marriage that had to be changed significantly.

The primary acting entity of the linked lives model is the individual person. In our ML3 implementation these are represented by agents af the agent type `Person` (figure 1, line 1-4). Each person is characterized by four attributes: its sex, its status, its current care need category and its socio-economic category. Additionally a persons behavior is influenced by its age. However, the age needs not be declared as an attribute in ML3. All ML3 agents have an implicit attribute **`age`**, that is zero at the time of the agent's creation and changes automatically when time passes. Finally, each agent is either alive or dead. Agents are alive when they are created and might die through events.

Married couples are also represented by agents (line 6). This way, we can model certain decisions, e.g., the decision to get a child, as a behavior of the couple, and not just as a behavior of one of the partners. The connection between a couple and the persons making the couple is done via a link. As the link declaration (line 7) declares, every couple is linked to two persons, the `spouses`. Every person might be linked to multiple couples. This way a person can not only access the couple that represents their current marriage, but also past ones. This is important, as we link the children of a couple to the couple, not the individual parents (line 8).

Finally, we have agents representing the houses and towns (line 10-13). Like persons, houses have an associated socio-economic category, that determines which persons can afford to live in that house. Each house is linked to the persons living in it and to the town it belongs to. The towns have attributes `x` and `y` for their position on the town grid.

Until now we have only described how a state of the model looks like. The dynamics of the model are described by stochastic rules. Every rule applies to agents of a certain type that are currently alive. It consists of three parts: the *guard*, the *rate* and the *effect*. Figure 2 shows one of the

```
1 Person
2 | ego.careNeedLevel < numCareLevels - 1
3 @ if ego.sex = "f" then femaleCareRate[ego.age]
      else maleCareRate[ego.age]
4 -> ego.careNeedLevel := min(ego.careNeedLevel +
      careTransition[random()], numCareLevels -
      1);
```

Figure 2: The rule for increasing care need with age.

rules of the model. This rule describes the degradation of a persons health, that increases their care need. The guard (line 2) specifies to which agents it applies. In this case it applies to all persons, who's current care need level is lower than the maximum care need level. The keyword **ego** refers to the agent the rule is applied to, similar to `this` in many object oriented languages. The rate (line 3) describes when the rule is executed. The rate's value gives the arrival rate of an inhomogeneous Poisson process that determines the times at which the rule is executed. In this case the rate depends on the agent's sex and age. It is given by two maps, `femaleCareRate` and `maleCareRate`, a special type of model parameter ML3 uses mainly for time series data. The maps define age intervals and rate values for each interval. The rule effect (line 4) describes what happens, when the rule is executed. Here it changes the agent's `careNeedLevel` by a random amount. The map `careTransition` specifies the distribution of the amount of change.

Similarly, we derived an ML3 rule for every event of the original model. Given the original Python code we could translate guard condition and rule effect directly to ML3, as the rule shall apply to the same agents and have the same effect as in that model. However, we need to take more care when it comes to the rate, as the rate determines the timing of events, and the original model has discrete time while time in ML3 is continuous. The original yearly transition probabilities have to be translated to transition rates, so that our continuous-time model approximates the timing of events the discrete-time model produces. In the discrete-time model the timing of an event is determined by a Bernoulli process with parameter p, the yearly probability of an occurrence of that event. In the continuous-time ML3 model event timing is determined by a Poisson process with parameter λ, the yearly event rate given by the rule's rate expression. However, when the step size of the Bernoulli process is sufficiently small, it can be approximated by a Poisson process with $\lambda = p$. So under the assumption that this is indeed the case we can simply use the values of the discrete-time transition probabilities as values for the continuous-time transition rates of the ML3 model.

While the translation of most of the model behavior was similarly straightforward as the above example, the match-making for marriage had to be changed in depth, because the original approach does not work in continuous time. In the original discrete time model every individual decides whether they intend to marry that year every year, with a probability depending on age and sex. Afterwards the men who intend to marry get matched to suitable women. Potential spouses are suitable if they do not have the same parents and are in a certain age range. However, in ML3 time is continuous, so the probability that two persons make their decision to marry at exactly the same point in time is zero. Therefore we introduced a marriage market [41]. When a person decides to marry, they will look for a suitable spouse at the marriage market. If they find one, they will marry them. Otherwise they will join the marriage market as a member and might get chosen when a suitable spouse decides to marry at a later time.

```
1  MarriageMarket();
2  marriageMarket:MarriageMarket[0-1] <->
       [0-]Person:members;
3
4  Person
5  | ego.status != "child", ego.single(),
       !ego.hasMarriageMarket()
6  @ if ego.sex = "f" then
       femaleMarriageRate[ego.age] else
       maleMarriageRate[ego.age]
7  -> MarriageMarket.all.only().addAndMatch(ego);
8
9  MarriageMarket.addAndMatch(?person : Person) ->
10   if (!?candidates.isEmpty()) then
11     ?couple := new Couple(),
12     ?couple.spouses := [?person, ?partner],
13     ego.members -= ?partner
14   else
15     ego.members += ?person
16   end
17 where ?candidates :=
       ego.members.filter(?person.canMarry(alter)),
18       ?partner := ?candidates.random();
```

Figure 3: The marriage market.

Figure 3 shows the implementation of this in ML3. We represented the marriage market itself as an agent. Therefore we defined a new agent type `MarriageMarket` of which exactly one agent will always exist (line 1-2). Persons are linked to this agent when they are currently members of the marriage market. A person's intention to marry is formed via a rule (line 4-7). It applies to all agents who are old enough to marry, currently unmarried and not already linked to the marriage market. The transition rates are again dependent on age and sex and were taken from the original model. In the rule effect the one existing agent of type `MarriageMarket` is retrieved and the procedure `addAndMatch` is called, so that the marriage market tries to match this person to a suitable spouse. The procedure itself is defined in line 9-18. The marriage searches for a suitable partner among its members (line 17-18). When it finds one, a new couple with the original person and the partner as spouses is formed and the partner is removed from the marriage market (line 11-13). Otherwise the person is added to the marriage market as a member (line 15).

4. EXPERIMENTATION WITH ML3 AND SESSL

To conduct simulation experiments with ML3 models we implemented an ML3 binding for SESSL. The binding covers the basic features of SESSL experiments, such as choosing model file and simulation algorithm, setting simulation stop conditions and replication numbers, and configuring parallel

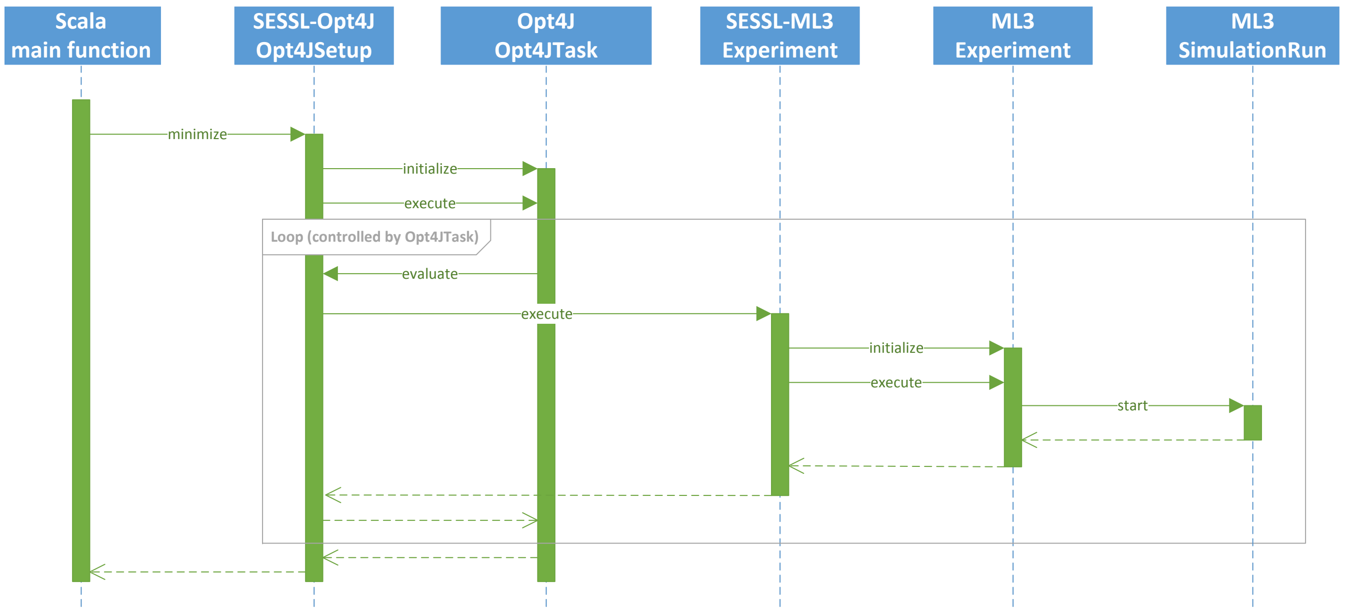


Figure 4: Call hierarchy in a SESSL experiment utilizing Opt4J [25] and ML3 (exemplified by the listing in figure 8). Both Opt4J and ML3 are proxied by the according SESSL bindings. The SESSL binding for Opt4J translates the optimization specification to an initialization and execution of the Opt4J package. To evaluate the target function, the SESSL-Opt4J binding executes the nested SESSL-ML3 experiment. This triggers the SESSL-ML3 binding, which in turn invokes the ML3 simulation package.

execution. In addition, we added some traits to the binding to address some experiment aspects that are specific to ML3. Similar as in other SESSL bindings to simulation packages, all specifications are translated to API calls of the ML3 package. A sketch of the call hierarchy of an experiment is shown in figure 4.

For the ML3 binding, we adopted a new concept of model parametrization. Previous SESSL bindings executed models with scalar model inputs. However, as ML3 models such as the linked lives model are aimed at describing demographic phenomena, many model parameters are maps. For example, for individuals the risk of dying depends on their age. The pattern of age-dependent event rates is typical for applications in demography. Consequently, we implemented a trait `ParameterMaps` that allows reading in parameter maps from .csv files.

To enable experimentation with the linked lives model, we addressed some model-specific aspects. For instance, the binding allows specifying an object that constructs the initial state of the model. The initial state of the linked lives model is generated by a tailored implementation, which is specified in the SESSL specification. We also implemented a trait `HealthCareCostObservation` that provides the means to invoke some model-specific observation code. This seamlessly integrates the domain- and model-specific experimentation aspects into SESSL.

The software implementations as well as model and experiment files described in this work will be made available on-line under an open-source license.

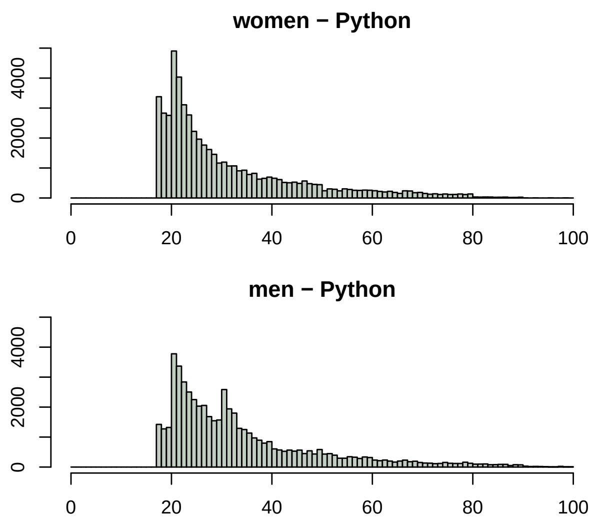
4.1 Experiments with the linked lives model

The experiments reported in this paper build on the preliminary work carried out on the original model [27], as reported in [33]. For the sake of comparability, the focus of experiments presented here, and thus the key output variable (Y), is the global cost of social care, expressed in British pounds, where for simplicity the hourly cost of providing care is assumed to be constant and equal £20. In [33], the four key input variables of interest included: the likelihood of aged parents returning home to live with their children (X_1), the retirement age for agents within the simulation (X_2), the hours of informal care provided by retired individuals to their family members (X_3) and the base probability that an agent transitions to requiring social care (X_4). The aim of the work as presented in [33] was to assess the uncertainty and global sensitivity of the simulation model output with respect to these four inputs based on a meta modeling approach. In the exercise with combined SESSL-ML3 experiments presented in this paper, the analysis is restricted to the first two inputs, X_1 and X_2 for the sake of transparency. In [33], Gaussian Process emulators [20] implemented in version 1.1 of the GEM-SA (Gaussian Emulation Machine for Sensitivity Analysis) software [19] were used to analyze the experiments statistically based on a meta-model. It is therefore of interest to see if with SESSL-ML3 we can replicate the effects of these two inputs reported in [33]. Hence, the key substantive question addressed in this paper is: what is the response surface of Y versus X_1 and X_2? To address that, a full factorial parameter scan of X_1 and X_2 is performed, and the effect of these two inputs on Y is assessed. For the scan, X_1 is being swept between 0 and 0.4, with step 0.02, and X_3 between 60 and 75 years, with step 0.5.

4.2 Experimentation with SESSL

```
 1  import sessl._, sessl.ml3._
 2
 3  execute {
 4    new Experiment with Observation with ParallelExecution with ParameterMaps {
 5      model = "./healthcare.ml3"
 6      simulator = NextReactionMethod()
 7      parallelThreads = -1
 8      replications = 5
 9
10      initializeWith(new HealthcareStateBuilder())
11      startTime = 1860
12      stopTime = 2050
13
14      fromFile("mortality.csv")("femaleDeathRate", "maleDeathRate")
15      fromFile("careTransitionRates.csv")("femaleCareRate", "maleCareRate")
16      fromFile("marriage.csv")("femaleMarriageRate", "maleMarriageRate")
17      fromFile("move.csv")("moveOutFromParentsRate", "singleMoveRate", "familyMoveRate")
18      fromFile("divorce.csv")("pastDivorceRate", "presentDivorceRate")
19      fromFile("careTransitionStep.csv")("careTransition")
20
21      set("numCareLevels" <~ 5)
22      set("ageOfAdulthood" <~ 17)
23      set("transitionYear" <~ 1965, "thePresent" <~ 2012)
24      set("minPregnancyAge" <~ 17, "maxPregnancyAge" <~ 42)
25
26      set("growingPopBirthRate" <~ 0.215)
27      set("steadyPopBirthRate" <~ 0.13)
28      set("coupleMovesToExistingHousehold" <~ 0.3)
29      set("moveTogetherRate" <~ 0.3)
30      set("agingParentsMoveInWithKids" <~ 0.1)
31
32      set("variableMoveBack" <~ 0.1)
33      set("ageOfRetirement" <~ 65)
34
35      observeAt(Creation("Couple"))
36      observe("maleMarriageAge" ~ expression("ego.husband().age"))
37      observe("femaleMarriageAge" ~ expression("ego.wife().age"))
38
39      withRunResult(result => CSVFileWriter(result, "variableMoveBack", "ageOfRetirement"))
40    }
41  }
```

Figure 5: SESSL specification for obtaining the marriage age distribution. In line 1, the SESSL core and the ML3 binding are imported. Line 4 contains the traits for the experiment to execute: we are interested in observation, parallel execution and reading in parameter maps. Lines 5 to 8 configure some basic experiment features, namely which model file, which simulator and how many parallel threads to use as well as how many replications to execute. The model initialization (using a tailored implementation) and the simulation start and end time are configured in lines 10 to 12. Lines 14 to 33 specify which parameters to read in as maps from .csv files, whereas others are set directly. The observation of the ages of women and men when marrying is configured in lines 35 to 37. Finally, line 39 states that the observed values shall be written to .csv files.



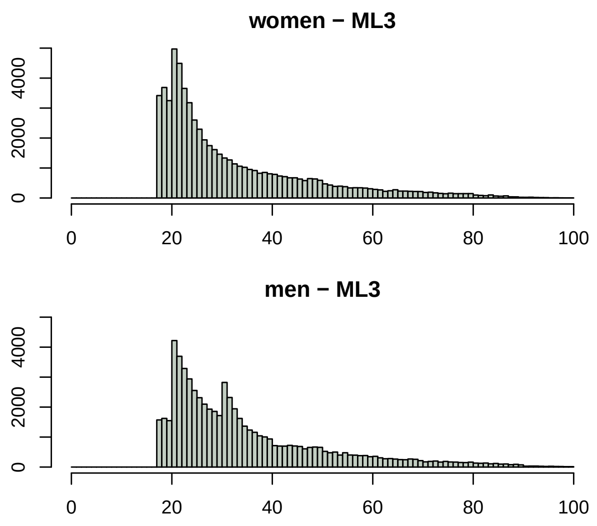


Figure 6: Histogram of the marriage age distribution for women and men in the original Python implementation and our ML3 implementation of the model. We are able to reproduce the marriage age pattern from the original Python model sufficiently well with a marriage market in the ML3 implementation.

As a first experiment, we conducted an experiment to cross-validate the ML3 model against the original Python implementation. Specifically, we wanted to confirm that our newly introduced marriage market produces the expected marriage age distribution. We produced the data with the listing in figure 5. The specification starts with some basic settings, such as model file, simulation algorithm, parallelization and replication number. After specifying the initial state construction and the start and stop time, a number of parameters is read in from files or set directly. Finally, the observation is specified: Here we are interested in the creation of new couple agents, particularly in the age of husband and wife of the newly formed couple. The observed data is written to a .csv file. In this experiment all parameters are fixed to the values extracted from the Python model, so there is only one simulation configuration. This configuration is executed with the specified number of replications — as the model did not exhibit a high variance, we chose a replication number of 5. The results in figure 6 show that the marriage age distributions indeed match.

In the second experiment, we explored the parameter space of the model by performing a full factorial parameter scan over two parameters. The listing in figure 7 shows the specification. The specification is largely similar to the previous one, with three differences. First, instead of setting all parameters to fixed values, we `scan` $21 \times 31 = 651$ model configurations. Second, we mixed in the trait `HealthCareCostObservation` into the experiment object, which enables the observation of the global health care cost. Third, the observation is configured to start in the year 2000. The gap between the simulation start time and the beginning of the observation is used as a warm-up phase of the model (as in the Python implementation). During this time, a realistic random population evolves from a simple initial state, consisting of a set of couples in the year 1860. The results of the experiment are visualized in figure 9.

The third experiment specification, depicted in figure 8, constitutes an optimization experiment. Such an experiment can be used to calibrate a model by finding a model configuration whose output most closely resembles some given reference data. Another application is shown here: we try to find a configuration for which the model output, the health care cost, becomes minimal. Instead of simply executing a specified experiment, we embedded the experiment in a `minimize` block. This makes the experiment part of the target function of an optimization algorithm, specified below the experiment. The optimization itself is performed by a particle swarm optimization algorithm in the Opt4J package, addressed through the Opt4J binding. The embedded experiment is performed repeatedly with parameter settings set by the optimization algorithm, thus exploring the parameter space and looking for an optimal parametrization of the model. Optimization exploits sequential experiment design, by refocusing the area of search iteratively. In this case, we are searching for a minimal weekly health care cost per tax payer in the year 2050. It is found for a maximal age of retirement of 75 (unsurprisingly) with a value of £85.72.

As most of the lines in the listings overlap, it is also possible to create a trait that contains these recurring lines. However, we omitted this additional technical indirection for clarity.

5. DISCUSSION

Whereas we showed in [36] that in principle all features of the linked lives model [27] can be described in ML3, we had no means to assess the model beyond a static check of its structural plausibility [2] at that time. In comparison to [36], additional efforts went into the model initialization which, although crucial for demographic models, we skipped in the earlier publication. In addition, we revised our marriage rules, as marriages between agents in continuous time require more effort than in discrete time [41]. By realizing a binding between the ML3 simulator and SESSL and replicating simulation results, we gained confidence that transforming the time-stepped agent model into the ML3 model,

```
 1  import sessl._, sessl.ml3._
 2
 3  execute {
 4    new Experiment with Observation with ParallelExecution with HealthCareCostObservation with
          ParameterMaps {
      // lines 5 - 30 as in listing 5
31
32      scan("variableMoveBack" <~ range(0, 0.02, 0.4))
33      scan("ageOfRetirement" <~ range(60, 0.5, 75))
34
35      observeAt(range(2000, 1, 2050))
36      observe("Y" ~ healthCareCost)
37
38      withRunResult(result => CSVFileWriter(result, "variableMoveBack", "ageOfRetirement"))
39    }
40  }
```

Figure 7: SESSL specification for a parameter scan experiment, based on the listing in figure 5. Here, the trait `HealthCareCostObservation` is mixed into the experiment object. This enables the observation of the global health care cost (line 36). In the lines 32 and 33 two model parameters are given as ranges to scan, meaning that all combinations of both parameter ranges are explored. The resulting data is shown in figure 9.

```
 1  import sessl._, sessl.ml3._, sessl.optimization._, sessl.opt4j._
 2
 3  minimize {(params, objective) =>
 4    execute(new Experiment with Observation with ParallelExecution with HealthCareCostObservation with
          ParameterMaps {
      // lines 5 - 30 as in listing 5
31
32      set("variableMoveBack" <~ 0.1)
33      set("ageOfRetirement" <~ params("ageOfRetirement"))
34
35      observeAt(range(2000, 1, 2050))
36      observe("Y" ~ healthCareCost)
37
38      withReplicationsResult(result => {
39        val runResults = result("Y").map(_.asInstanceOf[Double])
40        objective <~ runResults.sum / runResults.size
41      })
42    })
43  } using new Opt4JSetup {
44    param("ageOfRetirement", 60, 0.5, 75)
45    optimizer = ParticleSwarmOptimization(iterations = 10, particles = 5)
46    withOptimizationResults { results =>
47      println("Overall results: " + results.head)
48    }
49  }
```

Figure 8: SESSL specification for an optimization experiment, based on the listing in figure 7. The ML3 experiment is embedded in a `minimize` block, which makes the experiment part of the target function for an optimization algorithm. The optimization algorithm sets one of the parameters of the model (line 33) and receives the mean result of all simulation runs executed with that parametrization (line 40). Lines 44 to 48 contain the configuration of the optimization algorithm.

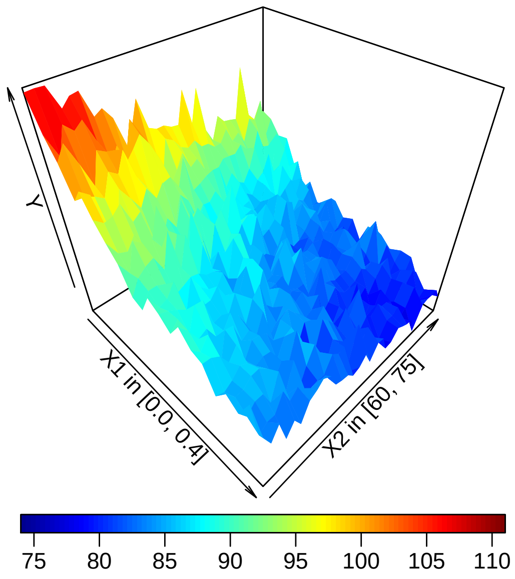


Figure 9: Response surface of the weekly social care cost per taxpayer in 2050 Y versus the likelihood of aged parents moving to their children X_1 and the age of retirement X_2.

an inhomogeneous CTMC, worked as intended. In the other direction, our experiments with the full model rather than an exploration based on statistical meta-modeling tools, such as Gaussian Process emulators, as was done in the earlier experiments [4], [33] and [16], confirm the findings that indeed the model is sufficiently benevolent for a meta-modeling approach, e.g., that outputs vary smoothly in response to changing a model's inputs. The little variance encountered in running the model kept us from using more sophisticated means to determine a suitable number of simulation runs in our SESSL experiments.

In realizing the SESSL binding to ML3 the observation required specific attention, as the observation shall not only be triggered by time but also by individual agents undergoing specific changes. The easy and clean invocation of custom, model- or domain-specific observation logic makes SESSL a powerful interface for diverse experimentation with simulation models. This is in line with current developments in adaptive collective systems [3], be this to introduce specific observation modules that consider the connectivity patters of social networks as proposed in [8] or spatial arrangements of agents [37] or monitor the spatio-temporal relations of agents based on definitions in spatio-temporal logic [26]. Similar reasoning applies for the initialization of the model. In agent-based models, diverse methods to create a random, but controlled initial population of agents exist [24]. Embedded domain-specific languages such as SESSL naturally support custom, ad-hoc implementations for the model initialization.

In the long term, model-specific code, for example for initialization and observation, might turn out to be partly applicable for other models as well. The corresponding code fragments can then be bundled into reusable experiment traits. This upstream movement of extensions enables a natural growth of SESSL's code base, as typical for many open source software projects.

Assessing how SESSL-ML3 helps to achieve the "gold standard" as defined in table 1, we see that SESSL supports diverse experimentation methods. In the examples we have shown a full factorial parameter scan, as well as a heuristically guided search in the parameter space using an optimization algorithm. Due to the easy extensibility of SESSL, additional methods can be integrated straightforwardly. Some advanced experiment design methods, such as Latin Hypercube sampling, are already available. In a similar fashion, methods to control the stochasticity of the simulation model during calibration and validation can be employed in experiments. For example, SESSL provides methods to compute confidence intervals of observations, whose width can serve as a replication criterion. Additional data processing steps (be this as a pre-processing or post-processing) are not that seldom, and here the concept of an internal domain-specific language allows an easy extension as well. For example, we integrated a LOESS smoother to filter out stochastic fluctuations before applying statistical model checking to an ML-Rules model of receptors [28].

In general, the number of tools and analysis techniques for agent-based models is steadily increasing, for some recent additions please consult [3]. Whereas the concept of internal domain-specific language allows an easy integration of these emerging and non-standard methods, it opens up the question whether such tools shall be integrated or rather loosely coupled with SESSL. Even, if specific cases, e.g. a pipe-lining of simulation and further analysis steps, suggest a loose coupling, SESSL would facilitate the setting up experiments and would make the first part of the data generating process explicit. Based on this, for example a meta-model can be learned, and as with any learning attempt the basis of learning is crucial for its success. In this case, the later part of the data producing process would rely on different means of documentation, for example on a scientific workflow tool in which SESSL is integrated. However, many methods can be seamlessly integrated into SESSL by providing suitable bindings to the respective tools and effectively be exploited (and evaluated) by specialized traits.

The thus increasing number of methods that become available allows to set up various experiments with agent-based models succinctly and reproducibly and, as SESSL-ML3 shows, to take specific experiment requirements of these type of models into account.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] B. Aparicio Diaz, T. Fent, A. Prskawetz, and L. Bernardi. Transition to parenthood: the role of social interaction and endogenous networks. *Demography*, 48(2):559–79, 2011.

[2] O. Balci. Verification validation and accreditation of simulation models. In *Proceedings of the 29th Winter Simulation Conference*, WSC '97, pages 135–141, Washington, DC, USA, 1997. IEEE Computer Society.

[3] M. Bernardo, R. D. Nicola, and J. Hillston, editors. *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer,*

Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20-24, 2016, Advanced Lectures, volume 9700 of *Lecture Notes in Computer Science*. Springer, 2016.

[4] J. Bijak, J. Hilton, E. Silverman, and V. D. Cao. Reforging the wedding ring: Exploring a semi-artificial model of population for the united kingdom with gaussian process emulators. *Demographic Research*, 29(27):729–766, 2013.

[5] F. Billari, T. Fent, A. Prskawetz, and B. Aparicio Diaz. The "Wedding-Ring": An Agent-Based Marriage Model Based on Social Interactions. *Demographic Research*, 17:59–82, 2007.

[6] F. Billari and A. Prskawetz, editors. *Agent-Based Computational Demography. Using Simulation to Improve Our Understanding of Demographic Behaviour*. Springer, Heidelberg, 2003.

[7] F. C. Billari. Integrating macro- and micro-level approaches in the explanation of population change. *Population Studies*, 69(Suppl.):S11–S20, 2015.

[8] L. Birdsey, C. Szabo, and K. Falkner. CASL: A declarative domain specific language for modeling complex adaptive systems. In *Proceedings of the 2016 Winter Simulation Conference*. IEEE Press, 2016.

[9] A. Coale. The demographic transition reconsidered. In *Proceedings of the International Population Conference*, pages 53–72, Liege, 1973. Ordina Editions.

[10] R. Ewald and A. M. Uhrmacher. SESSL: A domain-specific language for simulation experiments. *ACM Transactions on Modeling and Computer Simulation*, 24(2), 2014.

[11] T. Fent, B. Aparicio Diaz, and A. Prskawetz. Family policies in the context of low fertility and social structure. *Demographic Research*, 29:963–998, 2013.

[12] J. Grazzini and M. G. Richiardi. Consistent Estimation of Agent-Based Models by Simulated Minimum Distance. Technical Report 130, Laboratorioa Riccardo Revelli, 2013.

[13] V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback. The {ODD} protocol: A review and first update. *Ecological Modelling*, 221(23):2760 – 2768, 2010.

[14] A. Grow and J. Van Bavel. Assortative Mating and the Reversal of Gender Inequality in Education in Europe: An Agent-Based Model. *PLoS ONE*, 10(6), 2015.

[15] T. Hills and P. Todd. Population Heterogeneity and Individual Differences in an Assortative Agent-Based Marriage and Divorce Model (MADAM) Using Search with Relaxing Expectations. *Journal of Artificial Societies and Social Simulation*, 11(4 5), 2008.

[16] J. Hilton and J. Bijak. Design and analysis of demographic simulations. In J. van Bavel and A. Grow, editors, *Agent-Based Modelling in Population Studies: Concepts, Methods, and Applications*, pages 301–340. Springer, Dordrecht, 2016.

[17] B. Kamiński. Interval metamodels for the analysis of simulation Input-Output relations. *Simulation Modelling Practice and Theory*, 54:86–100, 2015.

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

[19] M. Kennedy. Description of the Gaussian process model used in GEM-SA. software manual, 2004.

[20] M. Kennedy and T. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society, Series B*, 63(3):425–464, 2001.

[21] A. Klabunde. Computational Economic Modeling of Migration. Technical Report 471, Ruhr University Bochum, Bochum, 2014.

[22] J. P. Kleijnen. *Design and Analysis of Simulation Experiments*. Springer, New York, 2008.

[23] D. Köhn and N. Novère. SED-ML - an XML formate for the implementation of the MIASE guidelienes. In M. Heiner and A. Uhrmacher, editors, *Proc. of the 6th International Conference on Computational Methods in Systems Biology*, pages 176–190. Springer, 2008.

[24] M. Lenormand and G. Deffuant. Generating a synthetic population of individuals in households: Sample-free vs sample-based methods. *Journal of Artificial Societies and Social Simulation*, 16(4):1–16, 2013.

[25] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, pages 1723–1730, Dublin, Ireland, 2011.

[26] L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti, and M. Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In E. Bartocci and R. Majumdar, editors, *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*, pages 21–37, Cham, 2015. Springer International Publishing.

[27] J. Noble, E. Silverman, J. Bijak, S. Rossiter, M. Evandrou, S. Bullock, A. Vlachantoni, and J. Falkingham. 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 WSC 2012*, pages 93:1–93:12, 2012.

[28] D. Peng, T. Warnke, F. Haack, and A. M. Uhrmacher. Reusing simulation experiment specifications to support developing models by successive extension. *Simulation Modelling Practice and Theory*, 68:33–53, 2016.

[29] H. Rahmandad and J. D. Sterman. Reporting guidelines for simulation-based research in social sciences. *System Dynamics Review*, 28(4):396–411, 2012.

[30] M. Reiter, U. Breitenbucher, O. Kopp, and D. Karastoyanova. Quality of data driven simulation workflows. In *Conf. on E-Science (e-Science)*. IEEE, 2012.

[31] J. Ribault and G. A. Wainer. Using workflows and web services to manage simulation studies (WIP). In *2012 Spring Simulation Multiconference, SpringSim '12, Orlando, FL, USA, March 26-29, 2012, Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, page 50. SCS/ACM, 2012.

[32] S. Rybacki, S. Leye, J. Himmelspach, and A. M. Uhrmacher. Template and frame based experiment workflows in modeling and simulation software with WORMS. In *Eighth IEEE World Congress on Services, SERVICES 2012, Honolulu, HI, USA, June 24-29, 2012*, pages 25–32, 2012.

[33] E. Silverman, J. Hilton, J. Noble, and J. Bijak. Simulating the cost of social care in an ageing population. In *27th European Conference on Modelling and Simulation, Aalesund*. ECMS, 2013.

[34] J. van Bavel and A. Grow, editors. *Agent-Based Modelling in Population Studies: Concepts, Methods, and Applications*. Springer, Dordrecht, 2016.

[35] D. Waltemath, R. Adams, F. Bergmann, M. Hucka, F. Kolpakov, A. Miller, I. Moraru, D. Nickerson, S. Sahle, J. Snoep, and N. Le Novere. Reproducible computational biology experiments with SED-ML - the simulation experiment description markup language. *BMC Systems Biology*, 5:198, 2011.

[36] T. Warnke, A. Steiniger, A. M. Uhrmacher, A. Klabunde, and F. Willekens. ML3: A Language for Compact Modeling of Linked Lives in Computational Demography. In *Proceedings of the 2015 Winter Simulation Conference*, pages 2764–2775. IEEE Press, 2015.

[37] T. Warnke and A. M. Uhrmacher. Spatiotemporal Pattern Matching in RoboCup. In M. Klusch, R. Unland, O. Shehory, A. Pokahr, and S. Ahrndt, editors, *MATES*, volume 9872 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 2016.

[38] M. Weidlich, G. Decker, A. Großkopf, and M. Weske. BPEL to BPMN: The myth of a straight-forward mapping. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 265–282. Springer, Berlin, Heidelberg, 2008.

[39] C. Werker and T. Brenner. Empirical Calibration of Simulation Models. Technical report, Max Planck Institute of Economics, 2004.

[40] U. Wilensky. Netlogo, 1999.

[41] S. Zinn. A Mate-Matching Algorithm for Continuous-Time Microsimulation Models. *International Journal of Microsimulation*, 5(1):31–51, 2012.