Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

FOUR YEARS OF NOT-USING A SIMULATOR: THE AGENT-BASED TEMPLATE

Dominik Brunmeir
Martin Bicher
Niki Popper

Institute for Information Systems Engineering
TU Wien
Favoritenstraße 9-11
1040, Vienna, AUSTRIA

Matthias Rößler
Christoph Urach
Claire Rippinger
Matthias Wastian

dwh Simulation Services
dwh GmbH
Neustiftgasse 57-59
1070, Vienna, AUSTRIA

ABSTRACT

With steadily increasing performance of computers, agent-based modeling has evolved from an analysis method for qualitative phenomena to strategy for quantitative decision support. With this orientation, however, the modeler faces new challenges during implementation. In particular, an appropriate simulation tool must feature the combination of data and model flexibility, process reproducibility, performance and portability. While existing simulators often do not sufficiently cover these features, it is also not sustainable to generally implement models from scratch. In this work, we want to present the idea of simulation templates as a compromise between the two strategies. We show, on the example of our Agent-Based Template and two use cases, the importance of the described challenges and how the simulation template concept supports solving them. We aim to generally promote the idea of developing a customized template, which, as a layer between simulator and from-the-scratch implementation, combines the advantages of both approaches.

1 INTRODUCTION

With the steadily increasing performance of computers, agent-based models (ABM) have evolved from an analysis tool to investigate qualitative phenomena to an approach for quantitative decision support. In these situations, the focus is no longer on the emergence and understanding of complexity, but on the most accurate microscopic representation of reality, the precise determination of the parameter values, and the quantification of outcomes and uncertainty. The trend towards quantitative use of ABMs has been observable for some time, but was undoubtedly fueled by the COVID-19 crisis. During this time, many research groups (e.g., Müller et al. 2021; Hoertel et al. 2020) used ABMs for epidemiological forecasting and scenario computation because, unlike classical compartmental models, many relevant aspects of the pandemic and pandemic management, such as contact tracing, school closures, virus variants, or immunity waning, can be modeled directly. Implementing an ABM for a fully integrated simulation application (compare with Popper et al. 2022) gives rise to multiple challenges. A simulation tool or software used for the implementation should support in these matters. Although the details and depth of these challenges can differ a lot between ABM research groups, they can, on the top level, be summarized into the five key areas shown in Table 1.

The most common strategies for the implementation of ABMs are (1) *from-the-scratch* using a fundamental programming language or (2) using an existing *simulator* or *simulation framework*. The first strategy is clearly the most flexible and portable one, but is neither efficient nor sustainable: Every algorithmic component, from random-number-generator to event-queue has to be implemented or integrated manually.

979-8-3503-6966-3/23/$31.00 ©2023 IEEE

Table 1: Challenges to an ABM simulation tool, when used in quantitative decision support projects.

Data Flexibility. The tool must offer methods to load and export potentially large amounts of data and must provide interfaces to various data structures such as CSV tables, JSON structures, domain specific XML files, or even SQL databases.
Model Flexibility. The tool must provide sufficient flexibility to implement all possible types of ABMs a modeler could think of. Moreover, the tool must support integrated use of the simulation within a custom outer framework, e.g., a custom calibration or optimization routine.
Process Reproducibility. The tool must ensure reproducible processes. This includes data preprocessing, model parametrization, random number generation, Monte-Carlo simulation, and code administration.
Performance. The tool must be capable of dealing with a potentially huge number of complex agents which leads to high memory and computation time demands. Therefore, the tool must use computationally optimized algorithms and must be capable of exploiting all computational resources given, e.g., via parallel computing.
Portability. Products developed with the tool and the tool itself must be fully portable and must run platform-independent. The ported tool must remain editable and maintainable. It must allow to add interfaces to third party products and various forms to present results, like visualizations.

Providing well-structured and easily readable source code as well as maintenance and communication of such becomes a time-consuming task which furthermore complicates onboarding of new colleagues. For the second strategy, a huge and constantly growing number of simulators and toolkits are available, but this comes with different challenges. Using a third party tool can be very restrictive with respect to model flexibility, performance or portability. If problems arise on the simulator level, the programmer is stuck and needs to look for a workaround.

In the past 15 years the team around the author has worked in projects in which large scale ABMs were used for quantitative decision support. In this time various from-the-scratch implementations were made and multiple simulators and toolkits were evaluated and tested for large-scale ABMs. During this work, the team experienced the mentioned problems: First, almost all from-the-scratch implementations (using Java, Python and C++) were eventually archived. Only the original developer could have theoretically maintained or further extended said projects within reasonable time. Tested simulators include AnyLogic (Grigoryev 2012) and Netlogo (Tisue and Wilensky 2004), which were both experienced great for prototyping, yet difficult to apply for large-scale applications due to issues with computational performance. Since most of our applications require millions of complex agents, neither AnyLogic nor Netlogo qualifies as a standard modeling tool for our research group.

An intermediate solution would be beneficial, i.e., a strategy that combines the advantages of a flexible from-the-scratch implementation with the benefits from using a simulator. Our solution to this problem is a **Simulation Template (ST) for ABM** , i.e. a code skeleton consisting of various algorithms and classes and a set of conventions, which (a) can and should be fully customized to the needs of the modeling team and (b) supports all points listed in Table 1. This way, the modeler does not have to re-implement basic simulation components in every new project, but is still left with sufficient flexibility to write a program that meets all the criteria and challenges for their specific application. This code skeleton can either be newly developed or taken, as a whole or in part, from one of the many ABM toolkits.

In 2019, the team around the author started with the development of their own ST for ABM: **ABT**, short for **Agent-Based Template**. The two toolkits Mesa (Masad and Kazil 2015) or Repast Simphony (North et al. 2013) were properly tested for their applicability as a code skeleton. Due to quickly visible performance deficiencies of Python compared to Java for agent-based simulations, MESA was quickly ruled out. Repast Simphony was tested in early 2019 on the example of a hospital utilization model (https://www.dwh.at/en/projects/premiqamed/). The ABM capabilities of the toolkit were fully sufficient to implement the model. Yet, we had difficulties overcoming certain design decisions, which are completely

reasonable for a holistic simulator, but not as well-suited as a template or library. For our use cases, we needed, e.g., a less tight coupling of the various components. We also consider a leaner core component to be beneficial for distribution. After some internal discussion and weighing all the pros and cons, we began implementing our ABT from scratch.

In this work, we will generally discuss the key ideas and advantages of using a simulation template as a compromise between the usage of an existing simulator and an implementation from scratch. For this purpose, we also present ABT, our own simulation template, and discuss its ability to properly handle the challenges in Table 1 for two use cases.

2 METHODS

2.1 Simulation Template for ABM

In our understanding, a ST for ABM is a customized collection of implemented code pieces and conventions which together support solving the challenges which occur in the course of the implementation, the application and the maintenance of ABMs in a research group (see Table 1). In our definition, it consists of two parts: an algorithm and a convention part.

The first part is comprised of multiple, layered components to facilitate building ABMs. As basic building blocks, the components allow the modeler to reason about the intricacies of a model without ambiguity about behavior of the system in the simulation context. In our understanding of ABM, both tick-based as well as event-based time updates are allowed. Hence, these components should always include a **discrete-event-scheduler** which handles the event-loop and general events. The scheduling logic and its concrete implementation should be chosen according to the specific type of models and performance requirements of the modeling team. Furthermore, the template should provide an **agent-based component**. It provides a dedicated agent class and everything necessary for describing and running an ABM: a simulation class with an interface for configuration, utility functions for keeping record of intermediate ephemeral simulation recordings, and a wrapper class for efficiently generating pseudo random variables in various distributions. Finally, the template should include an **input-output component**, responsible for managing input and output of simulation runs, tracking of system variables, as well as statistical examination of the output data.

The second part of a template is a set of written or implemented conventions modelers have to comply when implementing models with it. Three main areas should be covered: **version control**, **code structure**, and standards for **model in- and output**. Together, they ensure that reproducibility standards are met and support onboarding of new members to the programming team.

2.2 Implementation of ABT

In the following, we will present our own simulation template implementation, the ABT, and explain how it meets the requirements of our modeling team. For brevity we focus on the original Java-based implementation for the algorithm part, but the same concepts were also applied to a C# based version.

Discrete-event component. ABT provides both a tick-based scheduling system and a datetime-based one. The tick-based system is limited to integer values to model discrete increments as simulation progress, while the datetime-based system is better suited for describing events in a more calendar-centric way. This allows us to schedule events, e.g., *next week* or *in a month*. Because of the performance penalties of describing time in such a way, concrete timestamps are evaluated on demand. This is achieved through the *TimeInstant* class, which wraps and consolidates the *Java.time* classes. To break ties on simultaneously occurring events, a priority system was implemented. The higher the priority of a given event, the earlier it is triggered. The ordering of events, which may be created dynamically during runtime, is taken care of in the *{Tick-, Time}Scheduler* classes. They are based on priority-queues which have a runtime performance of $\mathcal{O}(\log n)$ for enqueing and dequeing operations.

The base object is the *Entity* which by default just provides a unique id. To enhance their capabilities, ABT provides the *Listener* interface to allow it to receive and react to messages or events. We distinguish between simple point-to-point messages and scheduled events. Messages are sent to their destination and processed (*handled*) immediately. They are primarily meant for inter-agent communication. Messages provide a source, a destination, and a unique id. Events are derived from Messages. They may be scheduled and processed either as soon as possible or at a future point in time. Since events are meant to be trigger by the *Scheduler*, they need to be sortable and thus have attached a datetime or integer in addition to a priority.

Usually, we derive from the *BaseEvent* class for the different types of events our models need. This allows us to separate responsibility and encapsulate behavior in distinct events. Another benefit of this granularity, is that ABT is capable to filter by event-type (*tracing*) during runtime for logging or visualization purposes.

Agent-based component. Our agent-based component is a layer on top of the discrete-event component with more capabilities. Its central element is the *Agent* class which is derived from the *Entity* class and has more complex properties, such as a name or a position, and by default implements the *Listener* interface to handle messages and events. Processing of messages can be restricted to a specific event-type to implement a simple broadcasting mechanism.

We usually employ Monte-Carlo (MC) simulation and have to ensure reproducibility with respect to pseudo random number (PRN) generation. This is guaranteed through our *Simulation* class, which uses by default a Mersenne Twister based pseudo random number generator. To enable parallelization, each instance of the *Simulation* runs independently from one another. They are started through a *Factory* class, that ensures proper (PRN-)seeding and initialization of used agents. The intermediate results are recorded in a special tabular data structure (see *Recorder* description below). Note, that ABT does, by itself, not offer any implemented methods for parallelization within single ABM runs, but solely for the Monte Carlo simulation. This is due to the fact that in all prior applications, we were always capable of fully utilize all available kernels on our computation server by parallel Monte-Carlo runs. Therefore, we did not yet put any further efforts in parallelization of individual runs or GPU programming.

Input/Output component. As our models have a very heterogeneous need for data import, we implemented various helper classes for importing and exporting data. Most of them are based on the *jackson* library, see Young (2013), as it works well with various tabular and hierarchical data structures.

Additionally, we provide a fast internal tabular datastructure, called *Recorder*, for collecting output information from agents during runtime. The main consumer of this class the *Statistician* class which collects all *Recorder* instances and provides methods for automated aggregation and statistical analysis.

For storing the state of our simulations, including the recorder, we use object-serialization with JSON. This allows us to create checkpoints and resume the simulation at arbitrary times. All the classes in our template can be serialized and deserialized by default. This allows ABT to save and restore the state of the simulation which can be used to drastically reduce the runtime cost of performing exhausting parameter variation, optimization, or calibration processes.

Version control. Our workflow is centered around source code and text-based configuration files. Hence, ABT must be used with a version control software like *git* (Chacon and Straub 2014) to track the progress of implementation and keep model results reproducible during development.

Code Structure. Our projects that build upon ABT implement the following classes:

- Versions of an *Agent* class: They provide the properties and the state of the model agents. Every *Agent* must extend a *BaseAgent* class provided by ABT.
- Versions of an *Event* class: They are typically used to describe the actions an agent takes at a certain point in time, but can also be used for administrative reasons such as time-ticks for output recording. Every *Event* must extend a *BaseEvent* class provided by ABT.
- *Config*: Configuration of the simulation experiment, such as general setup (e.g., name, random number seed, ...), general simulation properties (start-date, end-date, scale, ...) and scalar pa-

rameters and paths to parameter files. Typically this class is filled by a configuration file in JSON format (see below).
- *Simulation*: Responsible for initializing the agents. Also the central authority with respect to time-tracking, event scheduling, and recording of intermediate data. Must extend a suitable base class provided by ABT.
- *Run*: Starts the simulation experiment, takes care of instantiating the *Simulation* instances for parallelized MC simulation, and is responsible to process the results further as soon as all Simulation runs are finished. The *Run* routine is the main entry point for the program.

This basis allows us to cover many use-cases by default, while still retaining enough flexibility to adjust our models and data interfaces on actual demand. As all our classes are by default serializable, it was easy to write automatic tests for verification and performance optimizations of the basic systems. For cases not covered by unit tests, we compared known output or known behavior of old simulations with newer iterations of ABT and so ensured no involuntary API breakage. It also allowed us to replace computational expensive parts of the simulation, for example, sampling of complex agents, with reading them from a file and so allowed us to iterate faster.

Model in- and output. By convention, all model input is specified by one configuration file in JSON format. This file includes experiment-specific variables such as random number seed, Monte-Carlo iterations and scenario-name, scalar parameter values and all file paths to additional parametrization files. The usage convention of ABT suggests that the model output of any experiment is saved into a new folder labeled by a timestamp of the time it was started. The folder must include all results of all individual simulation runs collected by the *Recorder* classes, all statistics created by the *Statistician* class, as well as a full copy of the configuration file. This convention ensures that results are not unintentionally overridden, fully transparent, and easily reproducible.

3 CASE STUDIES

We will furthermore present two projects, both long-term decision support projects of TU Wien and dwh GmbH together with different stakeholders, in which a comparably large and complex ABMs were developed based on ABT. The models are fully validated, parametrized with real data, and are still maintained, enhanced and used for counseling (as of Spring 2023).

The two simulation models are fundamentally different: They (a) originate from different fields of application, (b) are based on different time update strategies, (c) use different data structures and interfaces for parametrization and calibration, and (d) have entirely different forms of result and model usage.

3.1 RailwaySim

In railway traffic, locomotives, usually called *traction units*, are the most valuable component. Hence, the *circulation plan*, the plan defining which unit is assigned to carry which train between which stations, must be planned properly to avoid overheads. This leads to a multifactorial optimization problem which involves reducing the number of units and the idle distance, i.e., the covered distance without carrying any carriages, at the same time. On top, if the schedule is too rigorously optimized, it is more vulnerable to long propagation of delays. The latter was key motivation for the development of **RailwaySim** in 2017 together with company partner Austrian Federal Railways OEBB, see Rößler et al. 2018. Given a circulation plan, the model assesses the robustness of the schedule as feedback to planners by simulating all Austrian trains, locomotives, delays and delay propagation.

Due to the highly intertwined planning processes at OEBB, more and more aspects were identified to be tested for robustness regarding delays and delay propagation such as shift plans for drivers and personnel, energy consumption, concepts for interruption of operations, and long term and strategic planning of resources and operations. Therefore, in its current version, **RailwaySim** plays the role of a common core for various robustness assessment studies.

3.1.1 Model Concept

The original concept of RailwaySim was presented in Rößler et al. 2020. Components of the model are:

Infrastructure is used and occupied during a simulation run. Infrastructure elements are *operational points*, *stations*, *tracks*, and *sections*, i.e., infrastructure that is directly used by trains, but can also include *bases* and *common rooms* for personnel or *garages* to repair traction units. **Resources** advance the simulation by executing a predefined task list. Resources include *trains*, *traction units*, or *train drivers* and *personnel*. **Tasks** have to be completed by and are assigned to (possibly multiple) resources. Each task is also assigned to an infrastructure element, which is occupied during its execution. Tasks include *train runs* and *stops*, *empty runs* of traction units, as well as operational tasks for drivers and personnel, that are planned during shift planning, like *driving trains*, *handovers*, or *paid* and *unpaid breaks* among others.

During simulation the execution of each task can be delayed by a **primary delay**, which is applied randomly for each simulation run using machine learning (ML) components. Those primary delays are propagated through the systems and lead to **secondary delays** in the following ways: (a) If a task is assigned to multiple resources, it can only be executed if all of these resources are ready. The delay of one of the assigned resources results in a delay of the task. (b) If infrastructure is already occupied and cannot support another task to be executed, the task is delayed until the infrastructure is available again.

The model is implemented in such a way that it can easily be extended to incorporate additional resources or outcomes such as the energy consumption during a task.

3.1.2 Parametrization Data

The model uses a variety of different data sources including infrastructure data and various schedule data: historic/actual freight/passenger traffic schedules for trains and traction units. In this context, the data for actual traction unit schedule is considered as the input of the model (see Section 3.1.3).

Data is provided in various formats including CSV-, JSON-, XML-files as well as SQL-databases. While using established standards, like railML (www.railml.org), the harmonization into a common data structure was challenging.

Historical schedule data is primarily used for validation and for training the ML model for prediction of accurate primary delays (Leser et al. 2019).

3.1.3 Model Usage

To assess robustness of a schedule, the model is run with two scenarios: a base scenario where availability of traction units is assumed to be always provided, and another scenario where the traction units must follow the circulation plan. A quotient, the so called *markup* value, between the regional/overall secondary delays from the two scenarios quantifies the regional/overall robustness of the plan. In this context, secondary delays are the ones resulting from delay propagation. Figure 1 shows an exemplary result visualization comparing planned and simulated schedules.

The stochastic nature of the model with randomly sampled primary delays makes it necessary to run it as parallelized MC simulations. Therefore, random number streams must be carefully synchronized between the two scenarios to compute unbiased markup values.

The model is used either with human (planners) in the loop or in a simulation-optimization setting with automated circulation planning optimization (Rößler et al. 2020).

3.1.4 Implementation with ABT

To be compatible with the overall software architecture of company partner OEBB the implementation was required to use the programming language of C# instead of Java. Being in full control of all components of ABT, a clone of the implemented core of the template in C# could be developed easily and the Java

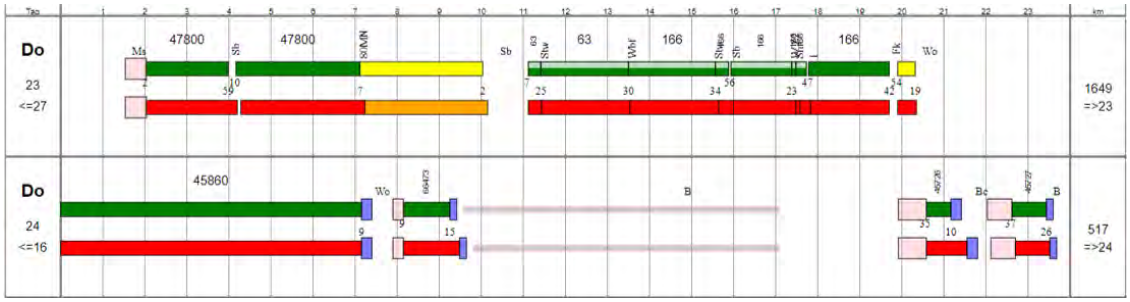

Figure 1: Visualization of simulation results of the RailwaySim model, that is used for robustness assessment of circulation plans. The figure shows the planned circulation plan in green and yellow and the corresponding simulation results in red and orange.

prototype of the model presented in Rößler et al. 2020 could be ported quickly. Verification-scenarios confirmed that the results of the ported model match the results of the original.

Since ABT is based on C#, we were able to directly incorporate the developed ML model for primary delays and generate the primary delay distribution for each task on runtime. Through this, we could also directly integrate changes in the ML-model due to updated data. This feature will be also used for the computation of energy consumption, as the energy demand of single tasks will be calculated based on a ML component.

The loose coupling of the components in ABT and the direct implementation in C# made the development of the integrated application for circulation planning, as presented above, possible. Furthermore it allows for a distributed and interchangeable architecture of various components where the implementation may be either changed during runtime or parts of the model may be processed outside of the simulation itself. The *tracing* framework and the serialization capabilities allow us to distribute these processes via Inter-Process-Communication (IPC) protocols, files or a database, as is the plan for other use cases.

The model development began with a small group of researchers, but the multiplicity of new use cases and research questions made it necessary to enlarge this team. While the new members were all skilled modelers and programmers, the template structure and conventions of ABT and the use of object oriented programming made communicating the inner workings of the model much more direct and therefore much easier.

The multiple use cases that need different features in the model make it necessary to have different instances of the model with different features. Basing all of them on ABT and the RailwaySim model makes it possible to share common features over multiple instances of the model and also to automatically integrate performance upgrades and extensions of one of the base models into all model instances.

3.2 SARS-CoV-2 Model

The second case study is an epidemic ABM which simulates the spread of SARS-CoV-2 in Austria. The model was developed to analyze the effects of various non-pharmaceutical interventions (NPIs) on the spread of the virus and the burden of the COVID-19 disease in Austria. The microscopic modeling strategy was chosen in favor of a compartment model because a modeled contact network was required to properly depict NPIs like school-closures or contact tracing. First results could already be generated and published in January 2020(Red, wien.ORF.at/Agenturen 2020). Key to this short development time were two older models which were used as basis: The first one simulates the spread of influenza and was implemented in Java in 2011 (Miksch et al. 2011). The contact module of this model was directly taken into the SARS-CoV-2 Model. The second model simulates the structure and dynamics of the Austrian population (GEPOC: (Bicher et al. 2018)). It was originally implemented in Python 3, but was ported to ABT in 2019. The two models were combined, adapted, and parameterized to the specific requirements of SARS-COV-2.

3.2.1 Model Concept

The SARS-COV-2 model is a comparably large ABM which consists of four different modules:

The **population module** (GEPOC) simulates the Austrian population with respect to age, gender, birth and death rates and regional distribution. Each inhabitant of Austria is represented as one model agent, leading to a model with approximately 9 million agents. The **contact module** models contacts between agents and the subsequent spread of the infections. Agents are grouped in households, schools, and workplaces, at which a given number of daily contacts occur with other members of these locations. Additional leisure-time contacts are generated between random agents according to their regional distribution. Figure 2 shows a snippet of the modeled contact network. The **disease module** simulates the "internal" progression of the infection in an agent. It decides, depending on the agent's immunity status, whether a contact with an infectious agent results in a transmission, and models the different stages of the disease, such as infectious period, symptoms, hospitalization, and gain and loss of immunity. The **policy module** handles all NPIs, as well as any other measure which can influence the spread of the disease. This includes lockdown measures such as school closures, but also quarantine or contact tracing regimes, and vaccination strategies.

The model implements a hybrid time update strategy. While policies as well as agent-agent contacts are modeled with discrete time steps (days), processes within individual agents, such as disease and immunity dynamics, are updated based on events. Most decisions such as the choice of contact partners, the risk of infection, or the duration of different disease stages are based on random distributions.

We refer to the supplemental material of Bicher et al. 2021 for more details on the model. Newer versions of the model specification were published on https://www.dwh.at/en/projects/covid-19/.

3.2.2 Parametrization Data

The model uses a variety of data from all sorts of data sources including demographic data (e.g., age and gender distribution, household configurations, birth and death rates, ...), contact data (contact location data, distribution of daily number of contacts, mobility data), virus data (transmissibility, latency time,...), immunization data (vaccine effectiveness, immunity waning distributions, ...), vaccination data (number and time of vaccinations) and epidemiological data from various national surveillance systems. The latter were used as reference for calibration of policy event parameter values. In total, the model relies on more than 30 different parameter files and about 50 additional scalar or vector valued parameters defined in the configuration file.

Often-changing data is maintained and updated using GIT and GIT-LFS. In particular, epidemiological data were kept on a secure server infrastructure due to privacy concerns and loading into the simulation is solved using a tailored aggregation interface.

With few exceptions data is given in form of CSV files and is loaded accordingly. Geographical information requires a parser for GEOJSON, a specific variant of a JSON files suited for geographical shapes, and GEOTIFF, an image-format for geographical raster images.

3.2.3 Model Usage

In general, the model is applied to answer what-if questions related to interventions yet also forecasts are possible. For most uses, the model needs to be first calibrated to the past epidemic spread. In this process, a MC simulation of the model is executed in the loop with a calibration heuristic to reproduce the daily confirmed cases. Afterwards, a snapshot of one of the simulation runs is created to get a simulated population which has the same properties regarding active or past infections and immunity as the Austrian population at the desired time point. This population can then be re-imported into the model and used for a multitude of research questions by taking a further look into the future.

This way, the model was applied for counseling various Austrian decision-makers and advisory boards between 2020 and 2022. These applications include the creation of weekly forecasts in the Austrian COVID Forecasting Consortium (Bicher et al. 2022), the evaluation of contact tracing (Bicher et al. 2021), the

analysis of undetected cases and herd dynamics (Rippinger et al. 2021; Bicher et al. 2022), and the evaluation of logistic strategies, such as the vaccine prioritization in 2021 (Bicher et al. 2022; Jahn et al. 2021; Jahn et al. 2022), and the generation of synthetic disease data, which can be shared with everyone without data security concerns for further epidemiological research (Popper et al. 2021).

3.2.4 Implementation with ABT

The portation of the underlying population model (GEPOC) from Python 3 to the Java-based ABT greatly improved the computational performance: Simulations became about five to ten times faster and required about one fifth of the original memory. In addition, the experimental setup and quality of the simulations were greatly improved by ABT reproducibility standards. Considering the continuously changing knowledge base in the COVID-19 crisis, the model and its implementation was continuously adapted and extended. The corresponding GIT repository has more than twenty branches and over thousand commits. Source data was updated daily using GIT and GIT-LFS as well.

Moreover, the contact module of the influenza simulation from 2011 could be incorporated almost unchanged into the new model thanks to the flexibility of ABT. Together with a new developed disease and policy module the SARS-CoV-2 Model finally became a large ABM: the source code, not counting source files from ABT, amounts to about 140 classes and more than 10,000 lines of code. The model is parameterized with about 30 CSV files and requires additional, more than 100 scalar parameter values to be set in the JSON configuration file. One simulation run with 9 million agents requires about 20-40GB of memory and takes 10-30s per simulated day. Calculation and calibration of the model is therefore performed exclusively on a dedicated computational server.

The optimized time-class was one cornerstone of improving the performance. Date queries are essential for pandemic simulations, since most external parameters (e.g., the timing of containment measures) are parameterized using a date-time format, but this format usually comes with computationally inefficient manipulations. The time class provided by ABT offers both date queries and time-optimized event scheduling. Another cornerstone was using individual discrete event schedulers for each agent, which was easily supported by the ABT Scheduler class. This way, events that only concern one agent, such as a change in disease severity or loss of immunity, can be handled completely inside the scope of the agent. This strategy both reduces unwanted interference between agents and the computational effort, since the internal events of 9M agents do not need to be scheduled and sorted into the global queue.

The visualization capabilities posed the only minus of the software switch: Since visualization libraries in Java are rather bulky, the ported model used an ABT interface to a Python 3 plot server, instead of implementing visualizations directly. More than 500 results with scenarios or forecasts were created, visualized and published in some format (scientific journal, newspaper, television, homepage,...) this way.

4 DISCUSSION

The two case studies shown highlight how the challenges presented in Table 1 arise in the respective application projects and how a simulation template supports solving them:

Data Flexibility. The two projects are characterized by specific requirements for importing and exporting data. These range from huge CSV and JSON files, specific XML or JSON formats (railML, GEOTIFF), geographic image files (GEOTIFF), to direct SQL database connections. The use of a customized template is optimal for the import of these formats, because besides the pre-implemented IO routines in ABT, new project specific parsers can easily be implemented and added to the method pool in the appropriate generic form. Furthermore, output standards of a template allow implementaion of direct interfaces to optimization algorithms, calibration algorithms an visualization tools.

Model Flexibility. The two models differ by very specific model mechanisms and time update strategies. For example, the RailwaySim requires runtime use of an ML method and the SARS-CoV-2 Model uses a hybrid time update strategy and agent-specific schedulers. The extensibility of the template

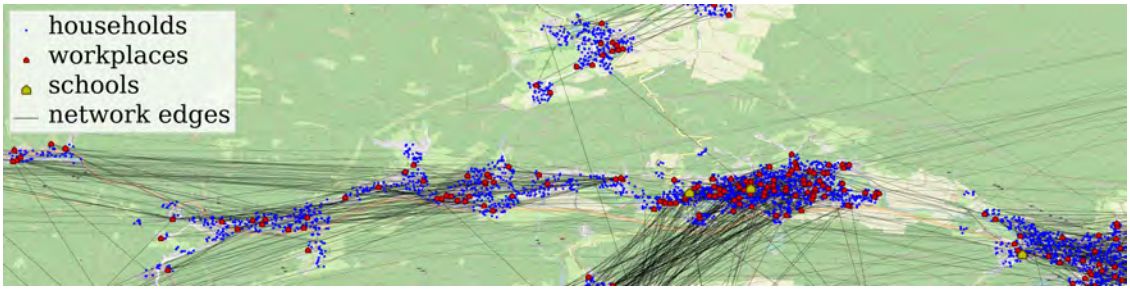

Figure 2: Visualization of the contact network in the SARS-CoV-2 Model. It shows the surrounding of the village Markt Piesting in Lower Austria with around 14×6 [km] (OpenStreetMap contributors 2017) and the modeled 12,287 inhabitants of the region together with their (locally reduced) contact network. Note, that all agents and locations are sampled based on regional distributions and are not taken from data directly.

approach eases the implementation of corresponding, special methods and therefore does not limit the modeler in the model design – at least not within the range defined by the project team. In the specific case of Java based ABT the sheer-endless availability of free third party libraries is helpful as well.

Reproducibility. In both projects the reproducibility of simulation results is of highest relevance, since these are used directly in the decision support. The main challenge here is to maintain reproducibility throughout the development phase of the model and under constantly changing model requirements, data and knowledge base. In ABT, the use of a configuration standard including the RNG seed of the experiment, and the required version control software for the code already provide a very high level of reproducibility. Together with a version management of the input files, full reproducibility is given.

Performance. The usability of both projects directly depends on the runtime of the model. In the RailwaySim it is decisive for whether the simulation is a bottleneck in the loop with the optimization, and in the SARS-CoV-2 Model it is crucial for the general computability of the model with its almost 9M agents. Many factors that negatively affect the performance of an ABM are found deep in the logic of the programming language (e.g., what type of list is used, or which number format is chosen). So they ultimately must be identified and optimized by the implementer of the model itself. Yet, a simulation template like ABT helps to harvest "low hanging fruits" with many default implementations, e.g., a heap-structure for discrete event queues or a KDTree for efficient spatial neighbor finding.

Portability. In particular, RailwaySim shows how special requirements for model portability can affect the project flow. The requirement to implement the model using C# rather than Java would completely preclude the use of an external simulator or a third party toolkit. However, since not only the model but also ABT was developed by our research group, both could be ported to C# with reasonable effort.

We have presented two case studies from the fields of transportation and epidemiology, in which the five mentioned challenges of quantitative ABM (Table 1) became clearly visible. A variety of other recent projects, also implemented with ABT but not described in detail here due to space limitations, also show that the same challenges can be found in other application areas: Healthcare, referring to a project to evaluate MMR and polio vaccination rates (Miksch et al. 2017), logistics, referring to a project to analyze bottlenecks in election processes (Weibrecht et al. 2021), or building infrastructure, referring to a project to optimize utilization in hospitals (Dwh GmbH 2019).

ABT is our customized simulation template and our answer to the challenges defined in Table 1. It has turned out highly useful in all agent-based simulation projects since 2019, it meets our expectations and needs of data and model flexibility, reproducibility, performance and portability, and it lies in our hands to adjust ABT to potential new needs in the future.

The idea of a template poses a third layer between using simulators, sometimes stiff and inflexible, and implementing models from the scratch, which may potentially imply reinventing the wheel. A layer, which

can be fully customized and may range from simple guidelines for code structuring to a huge toolkit of optimized classes and methods for various tasks related to implementation, parametrization and validation of conceptual models. A simulation template can ensure compliance with scientific standards by providing sound, uniform implementation of essential algorithmic components and conventions for all models based on it, while combining the onboardability features of simulators, i.e., that new members can be added to the project team easily, with the flexibility of a from-the-scratch implementation. Experience from multiple case studies, two of which were presented in this work, shows that the development of a simulation template causes overheads for a research group that pays off quickly.

ACKNOWLEDGMENTS

This research was funded in part by the Vienna Science and Technology Fund (WWTF) and by the State of Lower Austria [Grant ID: 1047379/LS22071], and Austrian Research Promotion Agency (FFG) [892235]. FFG is the central national funding organization and strengthens Austria's innovative power. Green-TrAIn-Plan (FFG project number 892235) is funded by the Federal Ministry for Climate Protection, Environment, Energy, Mobility, Innovation and Technology (BMK) as part of the call for proposals AI4Green. www.ffg.at. This research was funded in whole, or in part, by the Austrian Science Fund (FWF) I 5908-G.

REFERENCES

Bicher, M., C. Rippinger, G. Schneckenreither, N. Weibrecht, C. Urach, M. Zechmeister, D. Brunmeir, W. Huf, and N. Popper. 2022. "Model Based Estimation of the SARS-CoV-2 Immunization Level in Austria and Consequences for Herd Immunity Effects". *Scientific Reports* 12(1):2872.

Bicher, M., C. Rippinger, C. Urach, D. Brunmeir, U. Siebert, and N. Popper. 2021. "Evaluation of Contact-Tracing Policies against the Spread of SARS-CoV-2 in Austria: An Agent-Based Simulation". *Medical Decision Making* 41(8):1017–1032.

Bicher, M., C. Rippinger, M. Zechmeister, B. Jahn, G. Sroczynski, N. Mühlberger, J. Santamaria-Navarro, C. Urach, D. Brunmeir, U. Siebert, and N. Popper. 2022, May. "An Iterative Algorithm for Optimizing COVID-19 Vaccination Strategies Considering Unknown Supply". *PLOS ONE* 17(5):e0265957.

Bicher, M., C. Urach, and N. Popper. 2018. "GEPOC ABM: A Generic Agent-Based Population Model for Austria". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 2656–2667. Gothenburg, Sweden: Institute of Electrical and Electronics Engineers, Inc.

Bicher, M., M. Zuba, L. Rainer, F. Bachner, C. Rippinger, H. Ostermann, N. Popper, S. Thurner, and P. Klimek. 2022, December. "Supporting COVID-19 Policy-Making with a Predictive Epidemiological Multi-Model Warning System". *Communications Medicine* 2(1):157.

Chacon, S., and B. Straub. 2014. *Pro git*. Apress. https://git-scm.com/book/de/v2.

Dwh GmbH 2019. "Capacity Planning for Hospital Development". http://www.dwh.at/en/projects/premiqamed/. accessed 21st April 2023.

Grigoryev, I. 2012. *AnyLogic 6 in Three Days: a Quick Course in Simulation Modeling*. Hampton, New Jersey: AnyLogic North America.

Hoertel, N., M. Blachier, C. Blanco, M. Olfson, M. Massetti, M. S. Rico, F. Limosin, and H. Leleu. 2020. "A Stochastic Agent-Based Model of the SARS-CoV-2 Epidemic in France". *Nature Medicine* 26(9):1417–1421.

Jahn, B., G. Sroczynski, M. Bicher, C. Rippinger, N. Mühlberger, J. Santamaria, C. Urach, N. Popper, and U. Siebert. 2022. "HPR171 COVID 19 Vaccination - How to Support Decisions on Vaccination Prioritization for New Variants of Concern?". *Value in Health* 25(12):S263.

Jahn, B., G. Sroczynski, M. Bicher, C. Rippinger, N. Mühlberger, J. Santamaria, C. Urach, M. Schomaker, I. Stojkov, D. Schmid, G. Weiss, U. Wiedermann, M. Redlberger-Fritz, C. Druml, M. Kretzschmar, M. Paulke-Korinek, H. Ostermann, C. Czasch, G. Endel, W. Bock, N. Popper, and U. Siebert. 2021. "Targeted COVID-19 Vaccination (TAV-COVID) Considering Limited Vaccination Capacities—An Agent-Based Modeling Evaluation". *Vaccines* 9(5):434.

Leser, D., M. Wastian, M. Rößler, M. Landsiedl, and E. Hajrizi. 2019. "Comparison of Prediction Models for Delays of Freight Trains by Using Data Mining and Machine Learning Methods". *Simulation Notes Europe* 29(1):45–47.

Masad, D., and J. Kazil. 2015. "MESA: an Agent-Based Modeling Framework". In *Proceedings of the 14th PYTHON in Science Conference*, edited by K. Huff and J. Bergstra, 53–60. Austin, Texas: Enthought, Inc.

Miksch, F., N. Popper, M. Bicher, K. Haar, and M. Paulke-Korinek. 2017. "Evaluation Of Measles Vaccination Coverage In Austria". *Value in Health* 20(9):A798–A799.

Miksch, F., C. Urach, N. Popper, G. Zauner, G. Endel, I. Schiller-Frühwirth, and F. Breitenecker. 2011. "PIN101 New Insights on the Spread of Influenza Through Agent Based Epidemic Modeling". *Value in Health* 14(7):A284.

Müller, S. A., M. Balmer, W. Charlton, R. Ewert, A. Neumann, C. Rakow, T. Schlenther, and K. Nagel. 2021. "Predicting the Effects of COVID-19 Related Interventions in Urban Settings by Combining Activity-Based modelling, Agent-Based Simulation, and Mobile Phone Data". *PLOS ONE* 16(10):e0259037.

North, M. J., N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko. 2013. "Complex Adaptive Systems Modeling with Repast Simphony". *Complex Adaptive Systems Modeling* 1:1–26.

OpenStreetMap contributors 2017. "Planet Dump Retrieved from https://planet.osm.org". https://www.openstreetmap.org. accessed 21st April 2023.

Popper, N., M. Bicher, F. Breitenecker, B. Glock, I. Hafner, M. Mujica Mota, G. Mušic, C. Rippinger, M. Rössler, G. Schneckenreither et al. 2022, December. "Methods for Integrated Simulation: 10 Concepts to Integrate". *Simulation Notes Europe* 32(4):225–236.

Popper, N., M. Zechmeister, D. Brunmeir, C. Rippinger, N. Weibrecht, C. Urach, M. Bicher, G. Schneckenreither, and A. Rauber. 2021, April. "Synthetic Reproduction and Augmentation of COVID-19 Case Reporting Data by Agent-Based Simulation". *Data Science Journal* 20:16.

Red, wien.ORF.at/Agenturen 2020. "TU-Forscher Simulieren Coronavirus-Verbreitung". https://wien.orf.at/stories/3032379/. accessed 21st April 2023.

Rippinger, C., M. Bicher, C. Urach, D. Brunmeir, N. Weibrecht, G. Zauner, G. Sroczynski, B. Jahn, N. Mühlberger, U. Siebert, and N. Popper. 2021. "Evaluation of Undetected Cases During the COVID-19 Epidemic in Austria". *BMC Infectious Diseases* 21(1):70.

Rößler, M., M. Wastian, A. Jellen, S. Frisch, D. Weinberger, P. Hungerländer, M. Bicher, and N. Popper. 2020. "Simulation and Optimization of Traction Unit Circulations". In *Proceedings of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. M. Roeder, and R. Thiesing, 90–101. Online-Conference: Institute of Electrical and Electronics Engineers, Inc.

Rößler, M., M. Wastian, M. Landsiedl, and N. Popper. 2018. "An Agent-Based Model for Robustness Testing of Freight Train Schedules". In *Proceedings of the MAS: The 17th International Conference on Modelling & Applied Simulation*, edited by A. Bruzzone, F. De Felice, C. Frydman, F. Longo, M. Massei, and A. Solis, 101–105. Budapest, Hungary: I3M.

Tisue, S., and U. Wilensky. 2004. "NetLogo: A Simple Environment for Modelling Complexity". In *International Conference on Complex Systems*, 16–21. Boston, Massachusetts: New England Complex Systems Institute.

Weibrecht, N., M. Rößler, M. Bicher, Š. Emrich, G. Zauner, and N. Popper. 2021. "How an Election can be Safely Planned and Conducted during a Pandemic: Decision Support Based on a Discrete Event Model". *PLOS ONE* 16(12):e0261016.

Young, T. 2013. *The Jackson Cookbook*. Leanpub. https://leanpub.com/jacksoncookbook.

AUTHOR BIOGRAPHIES

DOMINIK BRUNMEIR is PhD student at TU Wien. He is the lead developer of ABT. His email address is dominik.brunmeir@tuwien.ac.at.

MARTIN BICHER has a PhD in Technical Mathematics from TU Wien and is expert in ABM. He was head developer of the dwh/TU SARS-CoV-2 modeling team during 2020-2023. His email address is martin.bicher@tuwien.ac.at.

MATTHIAS RÖSSLER is researcher at dwh GmbH. He focuses on M&S in health care, logistics, and energy simulations using ABMs, ODE- and DAE-based models. He is working on his PhD thesis at TU Wien. His email address is matthias.roessler@dwh.at.

CHRISTOPH URACH is researcher at dwh GmbH who focuses on modeling and simulation in health care and statistical analysis. He was respondible for data processes in the dwh/TU SARS-CoV-2 modelling team during 2020-2023. His email address is christoph.urach@dwh.at.

CLAIRE RIPPINGER is researcher at dwh GmbH who focuses on modeling and simulation in health care and ABM. She was one of the main developers of the dwh/TU SARS-CoV-2 model during 2020-2023. Her email address is claire.rippinger@dwh.at.

MATTHIAS WASTIAN is senior data scientist at dwh GmbH. His scientific focus lies on machine learning as well as modeling and simulation in the fields of logistics, health care and energy simulations. His email address is matthias.wastian@dwh.at.

NIKI POPPER is research associate at TU Wien and CSO of dwh GmbH. His research focus lies on comparison of different modeling techniques. His email address is nikolas.popper@tuwien.ac.at.