View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by University of Essex Research Repository

International Journal of Microsimulation (2017) 10(1) 106-134

International Microsimulation Association

JAS-mine: A new platform for microsimulation and agent-based modelling

Matteo G. Richiardi

Institute for New Economic Thinking at the Oxford Martin School, University of Oxford
Nuffield College and Collegio Carlo Alberto
matteo.richiardi@spi.ox.ac.uk

Ross E. Richardson

Institute for New Economic Thinking at the Oxford Martin School, University of Oxford
r.richardson05@alumni.imperial.ac.uk

ABSTRACT: We introduce JAS-mine, a new Java-based computational platform that features tools to support the development of large-scale, data-driven, discrete-event simulations. JAS-mine is specifically designed for both agent-based and microsimulation modelling, anticipating a convergence between the two approaches. An embedded relational database management system provides tools for sophisticated input-output communications and data storage, allowing the power of relational databases to be used within an object-oriented framework. The JAS-mine philosophy encourages the separation of distinct concepts, objects and functionalities of the simulation model, and advocates and supports transparency, flexibility and modularity in model design. For instance, JAS-mine allows to store the list of regressors and their estimated coefficients externally to code, making it easy to change the specification of regression models used in the simulation and achieving a complete parallelisation between the tasks of the econometricians and those of the programmers. Moreover, tools for uncertainty analysis and search over the parameter space are also built in.

KEYWORDS: SIMULATION PLATFORM, MICROSIMULATION, AGENT-BASED, SOFTWARE, OPEN-SOURCE

JEL classification: C63, C88

1 INTRODUCTION

We introduce JAS-mine (Java Agent-based Simulation library - Modelling In a Networked Environment), a new Java-based computational platform that features tools for discrete-event simulations encompassing both dynamic microsimulation (MS) and agent-based (AB) modelling. With the aim to develop large-scale, data-driven models, JAS-mine brings real and simulated data together by facilitating the integration of real world data into simulation models. Object-relational mapping is used to embed a relational database management system, allowing the power of relational databases to be used within an object-oriented Java framework.

JAS-mine provides specific simulation tools, along with a template for simulation architecture design. In particular, JAS-mine is built around the idea that model development generally involves the task of several people, who should work in parallel, possibly building on pre-existing models and modules developed either by the same research team or by other teams. This is what the 'mine' in JAS-mine stands for. The motivation for this modelling approach is the recognition that the real bottleneck in computational modelling comes not from processor power but from the human element of designing and writing code. Hence, parallelisation in development, and not just parallelisation in execution, becomes crucial.[1] To minimise the time it takes for model developers to create and develop software projects, transparency, flexibility and modularity are to be preferred over brevity of the code and performance. This is achieved by keeping distinct concepts, objects and functionalities separate as much as possible. To this end, data representation and management is automatically handled by the simulation engine, allowing the modeller to focus on developing the behavioural algorithms and processes of the model. Moreover, JAS-mine supports the idea that the model developer should be given full control over modelling issues, whereas the platform should be responsible for technical issues.

The software follows the open-source paradigm, meaning that it is freely available for people to use, review and help develop further, thus encouraging the refinement of the platform over time. JAS-mine aims to use standard, open-source tools that are available in the open-source community whenever possible.

Classes of models that can be built using the JAS-mine platform include not only agent-based models, but static and dynamic microsimulations, involving either discrete or continuous time (events can be scheduled in regular or irregular time-steps), and can feature open or closed populations.

This paper is not a tutorial on how to use JAS-mine; for such information we refer the user to the extensive documentation that can be found online, and a detailed description of an implementation of LIAM2's Demo07 model in JAS-mine (Richiardi & Richardson, 2016).[2] Instead, this paper discusses the philosophy of JAS-mine, the type of computational problems it is designed to address, its architecture and features.

The paper is organised as follows: Section 2 provides a motivation for JAS-mine, its design philosophy and the issues it addresses; Section 3 describes important specifications of the platform; Section 4 highlights some key features; Section 5 discusses the possible modes of running JAS-mine; Section 6 presents performance characteristics of a demonstration model and Section 7 offers our concluding remarks.

2 MOTIVATION

2.1 Why *another* platform?

JAS-mine is specifically designed to provide tools for both dynamic microsimulation and agent-based modelling, anticipating the convergence of the two approaches (Richiardi, 2013).

Historically, AB models and microsimulations have followed different trajectories, with AB models focusing more on theoretical issues and MS models being more data-oriented, often featuring processes modelled as probabilistic regressions. In general, AB models are structural models with a primary concern on *understanding*, while microsimulations are reduced-form models geared towards *forecasting*. As data becomes more readily available and technology becomes increasingly sophisticated at handling such data, there has been an inevitable trend of convergence between AB and MS modelling styles, with AB models evolving to be more empirical in nature and MS models integrating interactions and feedback effects.

Agent-based and microsimulation models exhibit many of the same features and can be described as belonging to the same class of discrete-event simulations. Indeed, from a mathematical and computational perspective the two approaches are identical; they are recursive models in which the number and individual states of the agents in the system are evolved by applying a sequence of algorithms to an initial population. However, the differences in scope and perspective between MS and AB modelling has impinged on the structure of the computer models used within each community.

AB models lead naturally to an explicit object-oriented representation, while MS models are generally built around a database which is evolved forward in time. This has led to the development of simulation toolkits which are specific to each field, such as NetLogo (Wilensky, 1999), RePast (North et al., 2013) and MASON (Luke et al., 2005) for AB modelling, and Modgen (Statistics Canada, 2009), LIAM2 (De Menten et al., 2014) and JAMSIM (Mannion et al., 2012) for MS modelling to name just a few.

In particular, existing agent-based tools such as NetLogo and RePast are not designed for large-scale, data-driven modelling. Input and output (I/O) communications play only a secondary role, and the analysis and visualization of model outcomes are often mixed up with model structure. This hard coding and lack of a clear modular structure makes it difficult to perform design of experiments (DOE) on the model, hindering their use in large-scale, data-driven projects where modularity and efficiency are vital aspects in understanding the behaviour of the model.

On the other hand, existing microsimulation platforms such as LIAM2 and Modgen are designed with microsimulation structures in mind.[3] This generally imposes a programming style with a very strict, ad-hoc grammar and syntax. Not only can such demands end up being too much of a model design straight-jacket – especially for AB modelling – but they also represent a considerable investment for the model developer, who is required to learn an idiosyncratic language just to use the specific MS toolkit. One microsimulation platform that shares JAS-mine's emphasis on portability, scalability and open-source access, is the OpenM++ software (OpenM++, 2013). However, as OpenM++ is an implementation of the Modgen language, it suffers from similar issues to Modgen with regards to coding style and language.

Evolving out of the JAS project (Sonnessa, 2004) that dates back to 2004, JAS-mine was created to make the development of 'hybrid' AB-MS models easier and to allow researchers to use the same tools for both approaches, to exploit economies of scale in learning and coding.

Hybrid AB-MS models might have agents interacting locally, on the basis of local information, in the AB spirit. Behavioural rules are generally expressed as more or less complicated if-else statements. However, agents might also have some probabilistic transitions, as in Orcutt-type microsimulations. For instance, consumers might randomly receive leaflets from producers, select the best offer, and then shop from that specific seller; or they might be influenced by their acquaintances (who might themselves evolve endogenously). Either way, it is a one-to-one transaction between one consumer and one producer that eventually takes place, affecting the balance sheet of both parties. At the same time, the researcher might choose to model the individual working status according to some estimated transition model, without the need to specify the worker-firm interaction. Hence, hybrid AB-MS models allow to blend more structural (AB) and more reduced-form (MS) modelling approaches.

JAS-mine was designed specifically to give the modeller such flexibility, together with the tools needed to build large-scale, data-driven models. It leaves the model developer with full control over modelling issues, whilst taking care of the technical issues behind the scenes. It is written in the widely used Java programming language and should thus be readily accessible to a large population of programmers. Its unique combination of features distinguish it from all of the aforementioned platforms.

2.2 Why Java?

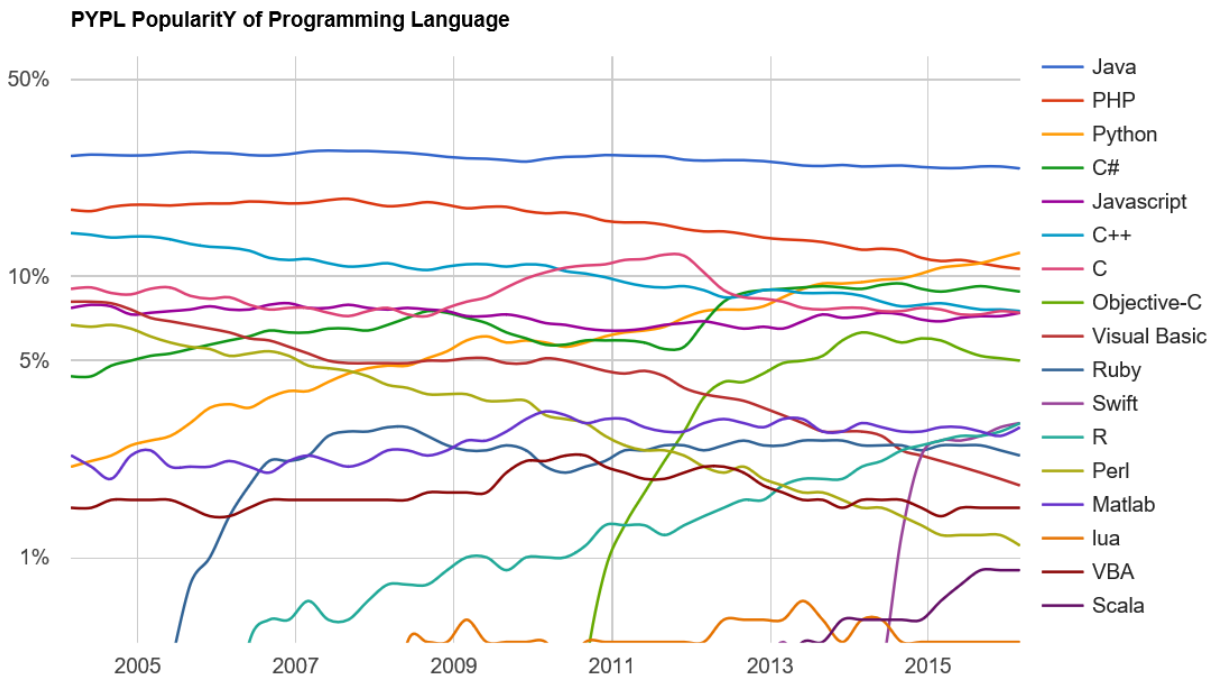
JAS-mine employs the most widely used and well-supported computing language available at this time – Java. Figure 1 from the 'PopularitY of Programming Language (PYPL) Index' illustrates the popularity of the most popular programming languages since 2004, quantifying the proportion of searches on Google for tutorials of specific programming languages. Across the whole period of time analysed, the Java programming language has maintained its lead as the most popular language with around 25% of all searches and, at the time of writing in 2016, has double the market share compared to the next most popular language, Python.[4]

This popularity enables the JAS-mine platform to benefit from the enormous contribution of human hours that have gone into developing a vast array of freely available, state-of-the-art Java tools. Furthermore, Java performs well in comparison with other widely used languages such as C (and C++) and Python, over a number of standard benchmarks, see Figure 2.

A user who develops an ability to code in Java in order to use, or even through using, the JAS-mine platform will obtain a widely applicable skill in high demand; such a skill is a valuable addition to any computational modeller's skill-set and will stand him or her in good stead regardless of whether their career is within academia or industry. Indeed, the learning curve for a new user of JAS-mine mainly rests on their ability to learn how to use the Java programming language.

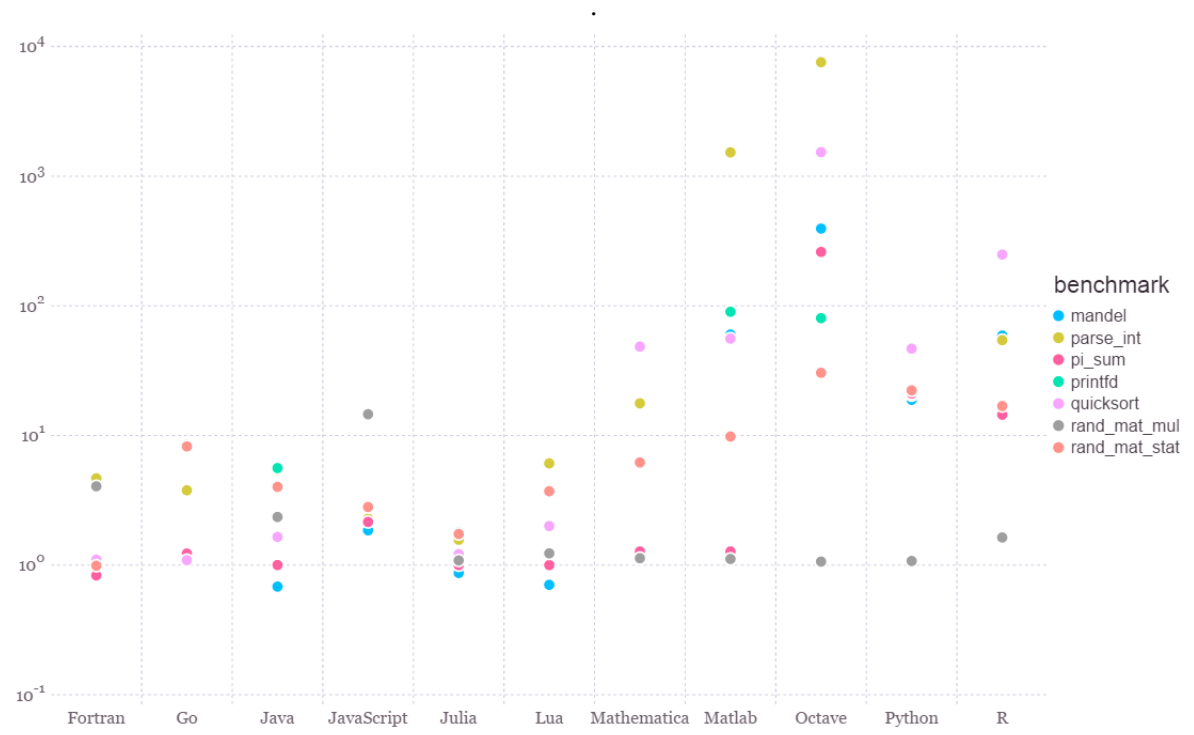
A direct competitor of Java – and its Scala variant – is Python, which is experiencing an increasing usage due to its flexibility.[5] In fact, Python is a multi-paradigm programming language, meaning that it can support both object-oriented and structured programming. Many language features also support functional programming

Figure 1: The most popular languages in the PYPL PopularitY of Programming Language Index



The index is created by analyzing how often language tutorials are searched on Google, from *http://pypl.github.io/PYPL.html*. Java has consistently been the most popular programming language over recent history and, at the time of writing in 2016, has 24.1% market share, double the amount of the next most popular language, Python.

Figure 2: Smaller is better: the benchmark times for operations implemented in a number of programming languages relative to C



The performance time of C is set to 1.0. Source: *http://julialang.org/benchmarks*.

and aspect-oriented programming, while specific extensions can provide support for other paradigms, including design by contract and logic programming. One of the biggest differences between Python and Java is the handling of variables. Java follows *static typing*, which requires to define the type of a variable when the variable is declared, and does not allow type changes in the program (though a variable can be cast into another type). In contrast, Python uses *dynamic typing*, which allows to change the type of a variable in the program. Dynamic typing is easier for the novice programmer; however, static typing can reduce the risk of bugs. Also, Java runs off the Java Virtual Machine, meaning that it is platform-independent. On the other hand, Python is a compiled language: the Python code is compiled into a code that the particular operating system used can understand. There are many Python interpreters: for instance, PyPy is a Python interpreter and just-in-time compiler which overcomes one of the main limitations of Python, namely speed, by compiling the program in C. However, PyPy only works with a subset of Python's libraries. For instance, it does not work with 'pandas', a popular Python library for data manipulation and analysis.

Apart from LIAM2, which is specifically designed for dynamic microsimulations in discrete time, other simulation platforms written in Python include the Python implementation of RePast, called RePast Py.[6]

An alternative to general purpose programming languages such as Java, Python or C++, or to platform specific languages such as Modgen (based on C++), LIAM2 (based on Python) or NetLogo (based on Scala/Java), is the use of all-purpose statistical packages, such as MATLAB, R or Stata. While examples of agent-based models written in MATLAB and microsimulation models written in Stata abound, to the best of our knowledge they make use of no tools to help with routine tasks in microsimulation modelling, such as the scheduling of the events, exchange of information between agents, etc. MATLAB has a block diagram environment for multi-domain simulation and Model-Based Design – Simulink – and a specific modelling and data analysis tool for discrete-event simulations, such as hybrid system models, agent-based models, state charts, and process flows. However, the tool (and MATLAB itself) is proprietary, and does not seem to have diffused into the field of social sciences. MATLAB can also run Dynare, a widely used platform for handling dynamic stochastic general equilibrium (DSGE) and overlapping generations (OLG) models. While Dynare also enables the development of models with heterogeneous agents, such models all remain in the rational expectations–optimal behaviour framework (or variations thereof), a framework which is often not shared by agent-based and microsimulation models.

On the other hand, specific microsimulation libraries exist for R, such as the package for Post-Keynesian Stock-Flow Consistent modelling developed by Antoine Godin.[7] With respect to MATLAB (or its open-source clone, Octave) and Stata, R is very flexible and powerful, and fully exploits the collaborative nature of open-source projects, with new and improved packages being continuously introduced. However, programming in R is often quite cumbersome and far from intuitive, with different packages often doing similar things in different ways and with different syntax. Moreover, R (as in MATLAB) is built around arrays (in particular matrices), while agent-based models and microsimulations benefit from an explicit object-oriented representation (see below), in particular when it comes to modelling the interaction between individuals.[8] Finally, R (as in MATLAB) can easily run into performance issues, especially if the modeller is not careful to avoid loops through an accurate vectorisation of the code, which in itself often leads to less intuitive and transparent syntax.

Finally, a very powerful simulation software is AnyLogic, a proprietary product of 'The AnyLogic Company',

which is mainly used for business applications.[9]

2.3 The JAS-mine Philosophy

Above all, JAS-mine stresses transparency, flexibility and modularity of the code, even (when there is a conflict) at the expense of brevity and performance.[10] The goal is to facilitate the model design and coding phase, minimising the time it takes model developers to create and develop large scale, data-driven discrete-event simulation projects.

JAS-mine's general principles maintain that model developers should be given full control over modelling issues; there should be no constraints on model specifications and no behavioural choices hidden in higher level functions, though the platform should assist in the implementation. In addition, the platform should take responsibility for technical issues such as setting parameters, managing the simulation schedule and I/O communications, the collecting of statistics, inspection and monitoring, and debugging.

Moreover, things that are conceptually distinct should be kept separate whenever possible, as discussed in detail in Section 3. For example, JAS-mine advocates the separation of the input data and parameters from the model code. This enables the parameters, and even the econometric and statistical specifications of regression processes in the model, to be changed without touching the code-base. This allows for efficient and flexible division of labour across time and space; users can collaborate and separately develop their own modules to be integrated into the overall project. This separation of distinct components encourages modularity, clarity and transparency.

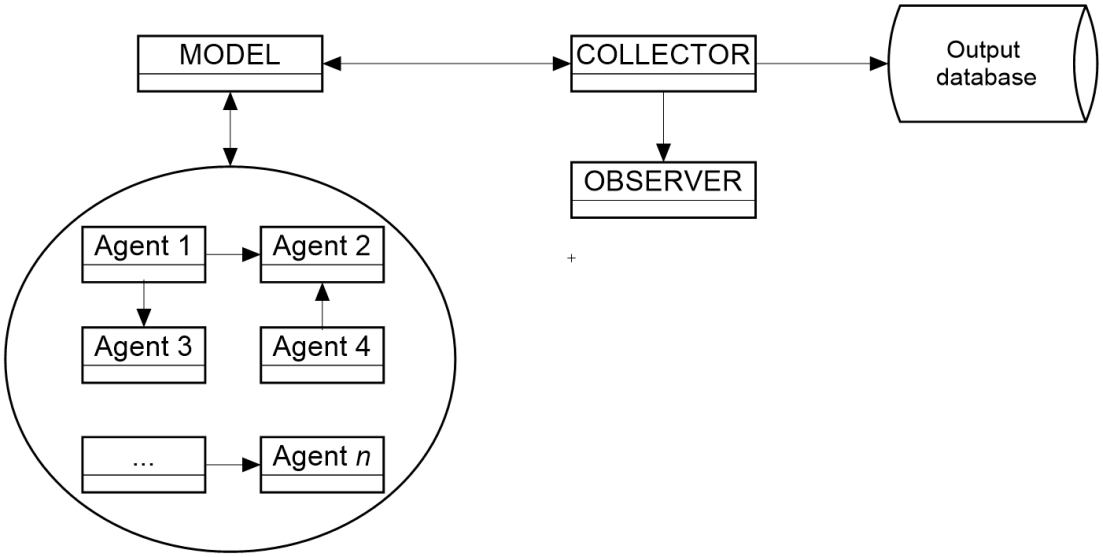
JAS-mine favours the use of Object-Oriented Programming (OOP) for its natural ability to represent the agents (individuals, households, firms, *etc.*) within AB and MS models. This programming paradigm further encourages well-structured code that is divided into packages and class hierarchies, supporting powerful computational modelling concepts such as encapsulation and inheritance (Luna & Stefansson, 2000; Gilbert & Terna, 2000).

JAS-mine aims to support the model building process by using transparent, well organized and documented functions. Being an open-source project, developers have access to all the JAS-mine source code should they wish to inspect and even refine it.[11] This helps to avoid the 'black box' nature that some modelling platforms suffer from, whilst encouraging further development of the platform. Moreover, JAS-mine inherits the object-oriented programming structure of packages and classes from Java, further facilitating the organization of code in a transparent, uncluttered manner. In addition, the JAS-mine website contains numerous tutorials, tips, demo models and the application programming interface (API), so that the model developer can find the necessary information to make the most of the JAS-mine tools.[12]

3 JAS-MINE SPECIFICATIONS

JAS-mine's design principles encourage adhering to a strict modelling discipline that maintains the separation between things that are conceptually separate. A clear distinction is made between objects with a modelling content, which specify the structure of the simulation, and objects which perform useful but auxiliary tasks, from enumerating categorical variables to building graphical widgets, from creating filters for the collection of

Figure 3: The structure of JAS-mine's Manager classes.



The Collector acts as an intermediate layer between the Model class (which controls the model specification such as agents, their interaction and environment) and the Observer (which displays information about the state of the simulation in the Graphical User Interface). The Collector aggregates information about the model objects (agents), and calculates any statistics needed either by the Model's objects (agents) themselves or for the Observer. The Collector can export the data to the output database or to csv files as required.

agents to computing aggregate statistics to be saved in the output database. This motivates the discussion in Subsections 3.1 and 3.2.

3.1 The Structure of a JAS-mine Project

A widespread paradigm used in the simulation of complex, agent-based models is Swarm (Minar et al., 1996), a system for organising discrete-event simulations where the computational objects of the simulation are partitioned into two separate groups:- those involved in describing the underlying model and those involved in observing what is happening. This mimics laboratory experiments in the natural sciences where the subject matter - the *Model* - is usually viewed as separate from the *Observer* performing the experiments.

The purpose of the Observer is to inspect the model's objects. Through the Observer, the state of the simulation can be monitored and graphically represented in real time, while the simulation is running. However, for the purpose of analysis and validation, the Observer alone may not be adequate because it implies the need to define in advance the aspects and aggregations on which to analyze the simulation outcome.

According to a different approach, the simulation is aimed exclusively at producing numerical outputs which can be analyzed in depth ex-post using ad-hoc statistical-econometric tools.

JAS-mine combines these two different approaches extending the *Model-Observer* paradigm so as to include an intermediate layer called the *Collector* that calculates statistical values and persists simulation modeling outputs in the database in the most transparent way, minimizing the impact on model implementation, see Figure 3. These layers are implemented within a JAS-mine project by objects called *managers*, who organise and manage agents within the simulation. We summarise the roles of each of the three separate managers below:

- The Model manager class deals mainly with specification issues, such as the creation of objects such as agents within the simulation, a description of relations and interactions between objects and the environment of the simulation, and defines the model's schedule of events.
- The Collector manager class introduced by JAS-mine takes care of recording and storing data from the model (data persistence). It builds the data structures and routines needed to collect data, and computes statistics required by the simulation objects and for analysis of the simulation run after completion. Its schedule specifies the frequency for sampling the agents, updating the aggregate statistics, and saving data into the output database.
- The Observer manager class builds and updates graphical widgets such as time-series plots or cross-sectional histograms in the JAS-mine graphical user interface to enable the user to inspect the state of the simulation in real time and monitor some predefined outcome variables as the simulation unfolds.

Each manager class contains code defining its own schedule of events, with the Model schedule arranging events that will occur between agents and their environment, the Collector schedule specifying when to calculate and record statistics of the underlying model, and the Observer schedule determining when to update the charts in the JAS-mine graphical user interface.

This three-layer methodological protocol allows for extensive re-use of code and facilitates model building, debugging and communication. Additionally, we highlight that there can be more than one type of each manager; for example two Model managers can be developed separately and easily assimilated into a JAS-mine project, as the JAS-mine simulation engine handles the aggregation of the two Models' schedules into the simulation engine's event queue. This allows for the creation of complex structures where agents of different Models can interact. Each Model is implemented in a separate Java class that creates the objects and plans the schedule of events for that Model.

For a detailed description of a JAS-mine project that demonstrates the separation of tasks into the Model-Collector-Observer structure, we refer the reader to Richiardi & Richardson (2016), which presents the porting of LIAM2's Demo07 demographic microsimulation model into JAS-mine.

3.2 Separation of Data and Code

JAS-mine favours the separation of data representation and management – which is automatically handled by the simulation engine – from the implementation of processes and behavioral algorithms, which should be the primary concern of the modeller.

In practicality, JAS-mine advocates the partitioning of data from the code-base, with all parameters and input tables stored either in Microsoft Excel files (.xls or .xlsx format) or in an input database. Although not a requirement, JAS-mine recommends that the only hard-coded parameters in the code-base are so-called '*GUI parameters*'. These GUI parameters are ones that the user wishes to directly set and possibly change during runtime; they are annotated as such (using the *@GUIparameter* Java annotation) to enable JAS-mine to recognise and display them in the graphical user interface (GUI). This results in quicker, more robust and more transparent model building, simplifying modular development and subsequent extension and modification.

For example, this separation allows the rapid iteration of model specification, as it is possible in JAS-mine to change not only the input data and parameters easily, but also to change a model's econometric or statistical specifications without changing any of the code: JAS-mine's regression package provides tools to import and inspect data from Microsoft Excel files, such that only the enumerated regression covariates and corresponding coefficients are used in the regression models. Thus, by removing a row containing a regression covariate and its corresponding coefficient from the Excel file, the JAS-mine regression object automatically removes this covariate from any related calculations, without the need to change any line of code in the code-base.

This separation of code and data greatly facilitates the evaluation of different econometric specifications and scenario analysis, in addition to the exploration of the parameter space. Moreover, this modular design allows for easier collaboration and efficient division of labour across time and space; an econometrician in one part of the world can develop the econometric (regression) model specifications, whilst a programmer in another part of the world writes the simulation code-base of the JAS-mine project in parallel.

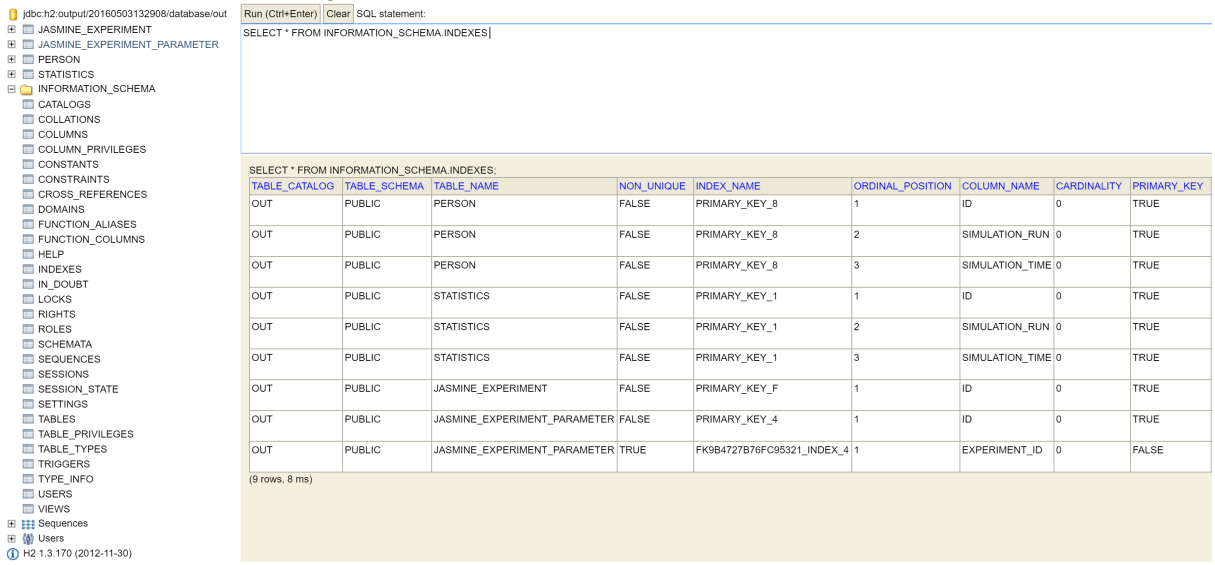
3.3 Input - Output Communications

A key feature of JAS-mine is its integration of input and output (I/O) communication tools within the modelling platform. By structuring the platform around a relational database management system (RDBMS), JAS-mine provides built-in utilities for communicating with underlying relational databases. These tools enable the user to import data from an input relational database and export data to an output relational database by writing just a single line of code for each operation.

Relational databases are an optimal way of storing vast amounts of data, potentially featuring complex inter-relationships. The statistical analysis of simulation output is possibly intensive in computing time, so time-constraints may limit such analysis in real-time, especially in large-scale applications. A common solution is to limit such analysis to a small subset of output variables, however this requires identifying the output of interest before the simulation is executed. If it is then decided that additional computations are necessary to better understand how the model behaves, the simulation has to be run again; the bigger the model, the more impractical this solution becomes. Relational databases make it feasible to keep track of a much larger set of variables and the relationships between agents in complicated simulation models, facilitating *post-mortem* analysis.

The benefits of having the simulation output stored as a relational database are larger when there are more object types in the model. For example, in an AB model where workers apply to vacancies issued by firms, there are four object types: workers, applications, vacancies, and firms, with each worker possibly applying to more than one vacancy, each vacancy possibly receiving more than one application, and each firm possibly posting more than one vacancy.[13] Moreover, attributes of each agent type can also be classified: for instance, all vacancies in the same industry/sector/area might share the same base wage, hours of work, paid holidays etc., as bargained between unions and firms. Rather than duplicating this information for each posted vacancy, we might store these characteristics in a separate table, to which each vacancy refers. A relational database keeps track of all the relationships between tables, as identified by primary and foreign keys. An alternative to using a relational database is to probe individual objects and save them in separate, unconnected tables, and indeed JAS-mine allows the user to follow this route and save the simulation outcome as separate text files (see Subsection 3.3.2 and

Figure 4: A screenshot from the Hibernate H2 console.



The console displays the information schema of a microsimulation model of labour force participation. There are two main tables: one containing panel data on individual agents (PERSON), and one containing time series aggregate data (STATISTICS). An additional table (JASMINE_EXPERIMENT_PARAMETER) contains the value of the model parameters (as set by the user through the GUI), while the JASMINE_EXPERIMENT table contains information about the specific run (run id and time stamp).

Section 6). Then, the relationship between the different tables can be inferred by looking at columns with the same name: for instance, the existence of a column named 'worker_id' in the WORKER, APPLICATION and VACANCY tables can be interpreted as workers applying to vacancies, and vacancies selecting one job applicant among all the received applications. However, there is nothing that tells the user that those different columns in different tables actually contain the same information: this knowledge must come from knowledge of the model structure. With the idea that the statistician analysing the model outcome can be different from the programmer coding the model, who may be different from the researcher specifying the model, storing all the relationships between attributes and agent types might be valuable.

The JAS-mine GUI contains a database explorer that links to a database console, allowing the user to inspect the input and output databases through Structured Query Language (SQL) style commands.[14] As an example, Figure 4 depicts the information schema of the embedded database of a JAS-mine project.[15]

Once the simulation has ended, the output database that JAS-mine has created can be loaded into the user's favourite statistical software (such as R, Stata etc.). This enables the user to employ all the powerful functionality of these programs to analyse the results of the simulations.

Each JAS-mine project can work with two databases: an *input* database and an *output* database. The input database can contain sets of model parameters and coefficients, and an initial population to be evolved forward in time by the simulation model. Any input values or population of agents could potentially be overwritten (i.e. in the *Model* class) after importing them if it were ever necessary to change the inputs. In addition, the Hibernate console provides tools that enable the user to construct input databases, e.g. using data from comma-separated values (csv) files.

The output database can hold data at both the individual agent (unit record) level and aggregate level, for example storing statistics that have been calculated within the model during the simulation. The output database

records the changes in the simulated population, either by sampling it at regular intervals in time or by recording individual events that happen to individual agents possibly at irregular times. The output archives the state of the system, including the initial period, and contains a copy of the parameters and coefficients used in the simulation, so as to avoid indeterminacies regarding how the data were produced. Thus, JAS-mine produces a copy of all the files in the input directory to store in the output directory, alongside the output database.

3.3.1 Relational databases in JAS-mine

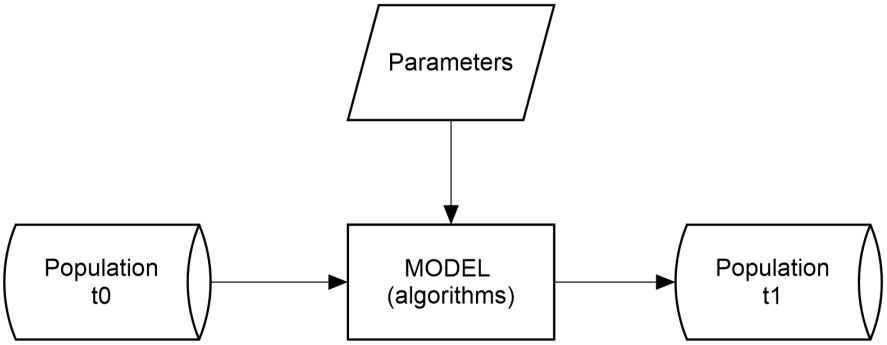
As we have seen, relational database structures are useful when wishing to store data from models with lots of inter-connected classes of agents with a variety of relationships, such as one-to-one, many-to-one or many-to-many connections. This is the case, for example, in simulations with complex many-to-many types of relationships.

Relational databases in JAS-mine contain a separate table for each entity (agent type). When constructing and storing data in an output database, JAS-mine produces a separate table for each Java class in the project that has been labelled with an *@Entity* Java annotation – we shall call such Java classes 'Entity Classes'. Each table in the database contains data from instances (the agents or 'objects' in Object-Oriented parlance) of the corresponding Entity Classes. A specific row in a table corresponds to an individual agent at a specific time in the simulation and is identified by a key containing numbers representing the agent's identity, the simulation time and simulation run (useful for identifying a run within JAS-mine's multi-run execution mode, see Subsection 5.3). Such a key corresponds to JAS-mine's *PanelEntityKey* data type (annotated with *@Id*), which must be declared in each Entity Class and be uniquely defined for each instance (agent) of the Entity Classes. Standard SQL queries can then be used to find a specific agent at a specific simulation time and simulation run in the database.

The output database records every data field that is defined in an Entity Class unless the field is annotated with the *@Transient* label. Basic data types such as individual numbers, strings, booleans and enumerated types are easily represented in the database, however only the reference field pointing to a more complicated Java Object would be stored for user-defined data types.

This introduces our discussion to the nature of relationships between agent types. It is possible for agents to have one-to-one, many-to-one and many-to-many relationships. In the labour market example of the previous section, the relationship between workers and vacancies is many-to-many, meaning that a worker can apply to many vacancies, and a vacancy can receive applications from many workers. Persisting a many-to-many relationship is complicated because the list of vacancies each worker has applied to is *a priori* of indeterminate length, as is the list of workers that have applied to any single vacancy. Persistence is then achieved by introducing an Application class that contains a pointer to the vacancy and the worker. Each application refers to one and only one link between a vacancy and a worker, and each link consists of one and only one application. The data that are saved in the database during the simulation refer to three different entities (workers, vacancies and applications) and are characterized by two different data structure (panel vs. population) however, thanks to the JAS-mine persistence engine, the appropriate keys are automatically added. This results in linked tables that can be easily manipulated in the subsequent analysis.

Figure 5: The structure of a microsimulation model.



Simulations can be viewed as data structures which evolve through time according to predefined rules and parameters.

3.3.2 Embedded Relational Database Management System via Object-Relational Mapping

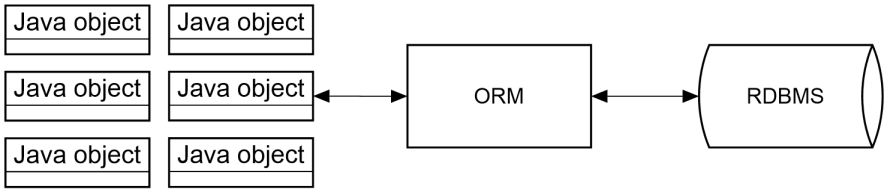
A natural way of coding systems of interacting agents, possibly belonging to different entities featuring hierarchical levels is through object-oriented programming (OOP). Indeed, this is the software paradigm best suited to represent and manipulate the sort of input data commonly found in AB and MS models such as population data. On the other hand, large-scale input and output data - especially in complex projects - are best stored in a relational database. Indeed, the traditional perspective of microsimulation modelling is that simulations are data structures which evolve through time according to predefined rules and parameters (see Figure 5). Database relational modelling however, is less intuitive than OOP and requires a specific language such as SQL to retrieve and modify the data.

JAS-mine overcomes the issues of interaction between the simulation and the I/O data by using an embedded RDBMS. An embedded RDBMS is a database management system which is tightly integrated with an application software that requires access to stored data, such that the database system is 'hidden' from the application's end-user and requires little or no ongoing maintenance. By default, JAS-mine uses the Hibernate (H2) database format, however other databases that support embedding can be used, such as Microsoft Access, Hypersonic SQL, Apache Derby, etc.[16] To change the database type, it is sufficient to reconfigure the *persistence.xml* file, which otherwise does not need to be modified. Also, by pointing the file *persistence.xml* to a database server it is possible to use the database in server mode, through a network interface.

Embedding is achieved using Object-Relational Mapping (ORM), a programming technique for converting data between incompatible type systems in OOP languages, see Figure 6.[17] ORM is used in JAS-mine to facilitate the integration of the object-oriented software system with a relational database (Keller et al., 1993). An ORM product (JAS-mine uses Hibernate) constructs an object-oriented interface to provide services on data persistence, while abstracting at the same time from the implementation characteristics of the specific RDBMS used.

Thus, all of the complex operations required to integrate the relational database management system into JAS-mine takes place behind the scenes. The ORM masks the complex activities involved in the creation, extraction, update and deletion of data behind simple commands, drastically reducing the amount of code required and removing a considerable burden for the model developer. These activities would have previously taken up a

Figure 6: Object-Relational Mapping



In JAS-mine the interaction between the simulation and the (input and output) data is achieved using Object-Relational Mapping (ORM), a programming approach that facilitates the integration of object-oriented software systems with relational databases. An ORM product (JAS-mine uses Hibernate) constructs an object-oriented interface to provide services on data persistence, while abstracting from the implementation characteristics of the specific relational database management system used.

large amount of the time required to write, test and maintain simulation models.

On the down-side, choosing an ORM paradigm introduces a software layer that impacts on performance, an aspect that is relevant to data-intensive applications like simulations. Translating the entity-relational model that is typical of a database into an object-based model requires additional activities that may slow down data upload, reading and exporting. Given the continuous increase in the speed and power of modern computers, we opt for a lean architectural structure even at the cost of slowing the simulation engine down. JAS-mine, however, does provide an alternative mechanism to export output data if the user wishes to increase the speed of the simulation. Instead of storing data in the output relational database, the user can choose to export data into comma-separated values files (csv), with a different csv file for each class of object exported. This can increase the speed of output substantially and is achieved simply by changing the value of two boolean arguments, either through the JAS-mine GUI or directly in the Collector class(es).[18] We explore the impact on the execution speed, of exporting data to the database and csv files in Section 6.

3.4 System Requirements & Database Size Limits

The only major requirement needed to use JAS-mine is the ability to run Java (version 7 or later) on a computer. JAS-mine has been tested on 32-bit and 64-bit Windows, Linux and Mac operating systems. The necessary random access memory (RAM) requirements to run a simulation successfully depends on the size (e.g. number of agents) and complexity of the simulation model, and the size of any input or output data involved. The minimum requirements to run Java 7 are a RAM of at least 128MB for 64-bit Windows XP and 64MB for 32-bit Windows XP, and around 124MB of hard drive memory to run on Windows.[19]

Development of JAS-mine projects can be facilitated using an integrated development environment, and we suggest using Eclipse Integrated Development Environment as we have produced a Plugin that can be used with Eclipse to help quickly set-up a JAS-mine project with the recommended file structure (see Subsection 4.6 for more information). Consequently, the system requirements in order to do this depend on Eclipse, which currently requires around 500MB of hard drive memory.

If using the default choice of database format – Hibernate (H2) – the size limits of any input or output databases handled in a JAS-mine simulation are determined by Hibernate's technology.[20] In particular, the database file size limit is 4 terabytes (TB) (when using the default page size of 2KB) or larger when using a higher page size. The maximum number of rows (records) per table is 2^{64}, and the minimum main memory required is around

1MB for each 8GB of database file size. There are no limits for the number of columns (attributes) in the database tables, other than those imposed by the memory and storage capacity of the computer.

4 KEY JAS-MINE FEATURES

We describe a number of useful features that JAS-mine provides to the model builder.

4.1 Simulation Time

JAS-mine allows great flexibility with regards to the time that an event can be scheduled. Whereas some microsimulation platforms such as LIAM2 only allow events to be scheduled at regular time-steps labelled by integers ('discrete-time'), time in JAS-mine is a continuous variable. This means that JAS-mine can handle complicated sporadic events that are scheduled at irregular time intervals, possibly sampled from a continuous probability distribution such as the exponential distribution to model inter-arrival times of events corresponding to Poisson processes. This flexibility is indeed required to implement the AB-MS hybrid model described in Section 6.

In addition, events that are scheduled for the same time can have their relative order specified using JAS-mine's scheduling methods[21]; this may be necessary to ensure strict causality in a simulation model. Dynamic scheduling is also possible within JAS-mine; events need not all be specified at the start of the simulation but can also be scheduled during runtime, for example by the agents themselves scheduling events that they will perform in the future.

4.2 Statistics

The JAS-mine release uses the Colt libraries for High Performance Scientific and Technical Computing developed at CERN, and the Apache Commons Math libraries in order to provide useful mathematical and statistical utilities.[22] The JAS-mine statistical library deals with the construction and update of cross-sectional and time series objects: statistics over a single agent (e.g. whether the agent is active or not) or a cross-section of agents (the activity rate) are overwritten on updating, but they can feed time-series objects which keep memory of past values. In addition, the overall population can be refined to a sub-population using filters to separate out agents that don't exhibit the required properties, such as a specific age or gender. The approach is fully modular: statistics can be computed on time-series objects (say, the maximum and minimum stock price in a given period of time), and stored for instance in other time-series objects. Furthermore, when a time-series object is updated, it automatically updates all the objects on which it is based.

For examples of how JAS-mine's statistical tools can be used, we refer the reader to both the documentation on the JAS-mine website and implementations in JAS-mine's demonstration projects.[23,24]

4.3 Regressions and Uncertainty Analysis

Sophisticated regression libraries allow a complete separation of regression specifications from the code. There is currently support for linear regressions, binary logistic, binary probit, multinomial logistic and multinomial probit regressions. The structure of the regressions can be delegated to the data stored in Microsoft Excel files (.xls and .xlsx), which contains both the name of regression covariates and corresponding coefficients for each covariate. During the simulation, the JAS-mine simulation engine will search for a particular definition of a regression covariate in the code and calculate its quantity for each agent that the regression applies to. The separation of regression specifications from the code mean that a regression covariate can be removed from the model simply by removing the covariate's corresponding row in the Excel spreadsheet. Furthermore, regression coefficients can be updated between simulation executions merely by changing the values within the Excel spreadsheet.

As utilised in Richardson et al. (2016); Richardson & Richiardi (2016), regression utility tools are available to facilitate the analysis of uncertainty in model parameters, pointing to the imprecision of the estimates and/or externally provided parameters (Bilcke et al., 2011).[25] One approach to deal with this uncertainty (Creedy et al., 2007) prescribes to bootstrap the regression coefficients of the estimated equations from their estimated joint distribution (e.g. multivariate normal in the case of multinomial probit regressions) with mean equal to the point estimate and covariance matrix equal to the estimated covariance. Bootstrapping needs to be performed only once, at the beginning of each simulation run: the entire simulation is then performed with the bootstrapped values of the coefficients. JAS-mine allows for a simple implementation of this 'brute-force' approach, by providing a bootstrapping method in the Regression library to be used within a multi-run execution mode.[26] The simulation is run many times, each using a different set of regression coefficients. The result is a distribution of model outcomes, around the central projections obtained with the estimated coefficients, as can be seen in Figure 7, taken from Richardson et al. (2016).

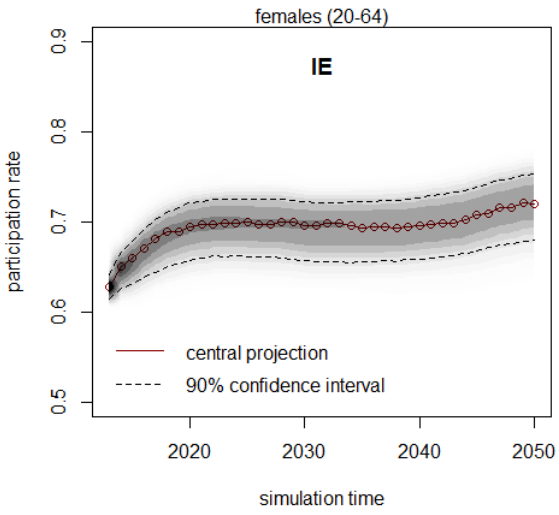
4.4 Alignment

Alignment is a technique widely used in dynamic microsimulation modelling to ensure that the simulated totals conform to some exogenously specified targets, or aggregate projections (Baekgaard, 2002; Klevmarken, 2002; Li & O'Donoghue, 2014).

Alignment is a way to incorporate additional information which is not available in the estimation data. The underlying assumption is that the AB or MS model is a poor(er) model of the aggregate, but a good model of individual heterogeneity: by forcing the microsimulation outcomes to match the targets in a way that is as least distortive as possible, the microsimulation model is left with the task of distributing the totals in the population. In general, the above assumption is very dangerous and unwarranted, and alignment should be looked at with great suspicion.

Nevertheless, a number of alignment algorithms are available from the JAS-mine libraries.[27] These include Resampling Alignment (Leombruni & Richiardi, 2006; Richiardi & Poggi, 2014), Sidewalk Alignment, Multiplicative Scaling Alignment, Sorting By the Difference between predicted probability and a random number

Figure 7: The effects of parameters uncertainty.



JAS-mine's regression utility tools is used to bootstrap regression coefficients during a multi-run execution of a simulation over 1000 runs. The chart was produced using kernel density estimation from analysis of the simulation's output database performed by the statistical software program '*R*'.

(SBD), and Sorting By the Difference between logistic adjusted predicted probability and a random number (SBDL). Descriptions of these alignment algorithms can be found in Li & O'Donoghue (2014).[28,29]

4.5 Matching

JAS-mine has specific tools contained within the *Matching* package, to perform matching between two collections of agents based on some specific criterion. The matching methods are called from outside the agents to be matched, for instance by the *Model* class. The simplest algorithm is a one-way matching procedure implemented in JAS-mine's *SimpleMatching* class, where the agents in one collection (e.g. females) choose to match with the agents in the other collection (e.g. males), who remain passive with regards to the matching process.

Matching is used to simulate the marriage between females and males within the population in the Demo07 demonstration model (Richiardi & Richardson, 2016).[30] More details and discussion can be found on the JAS-mine website.[31]

4.6 Extensions & Third-Party Solutions

The fact that JAS-mine projects are written in the Java programming language means that there are many state-of-the-art third-party solutions freely available for the modeller to use, not only within the code itself but also during the development of the code. For example, the Eclipse Integrated Development Environment (IDE) is available to use with JAS-mine; it features a wide variety of built-in development tools such as a powerful debugger, in-line help facility, refactoring tools and a Git version control system.[32] Furthermore, a vast collection of additional tools are available from third-party providers via the Eclipse Marketplace, such as a software profiler

that can be used to discover bottlenecks – the places in the code that take the longest time to execute – in order to aid the developer in making code run faster and more efficiently.

Moreover, a JAS-mine Plugin for Eclipse IDE exists that helps the model developer set up a standard JAS-mine project structure automatically within Eclipse.[33] This standard JAS-mine project features the recommended package structure and class names, and contains templates of the necessary classes with the required code in order to make the project ready for immediate execution. The project dependencies are handled by the Apache Maven Project, which automatically downloads the correct versions of files required by the project in order to run.[34] This enables the model developer to be instantly productive, as he or she can concentrate on writing the fundamental code to specify the model processes and agents' behaviour, rather than worrying about how to set up the JAS-mine project.

4.7 Simulation Scenarios

Once the input of a model scenario has been specified by the model developer (for example, in MS Excel files that hold the scenario parameters), such a model scenario can be executed at a later time by a user of the microsimulation model. The model developer can make it easy for a non-technical user to choose model scenarios by setting up dropdown boxes on the JAS-mine GUI that specify either the whole scenario or different components of the scenario. For example, the Labour Force Participation demo model has a dropdown box that allows the user to select which one of six possible countries to simulate, and the model has been built in such a way that the appropriate input Excel files containing the relevant parameters for the chosen country are automatically loaded by JAS-mine.[35] Another example is the Theoretical Health Inequality Model described in Section 6, where the GUI contains two dropdown boxes allowing the user to select the type of city scenario and the income inequality scenario.[36]

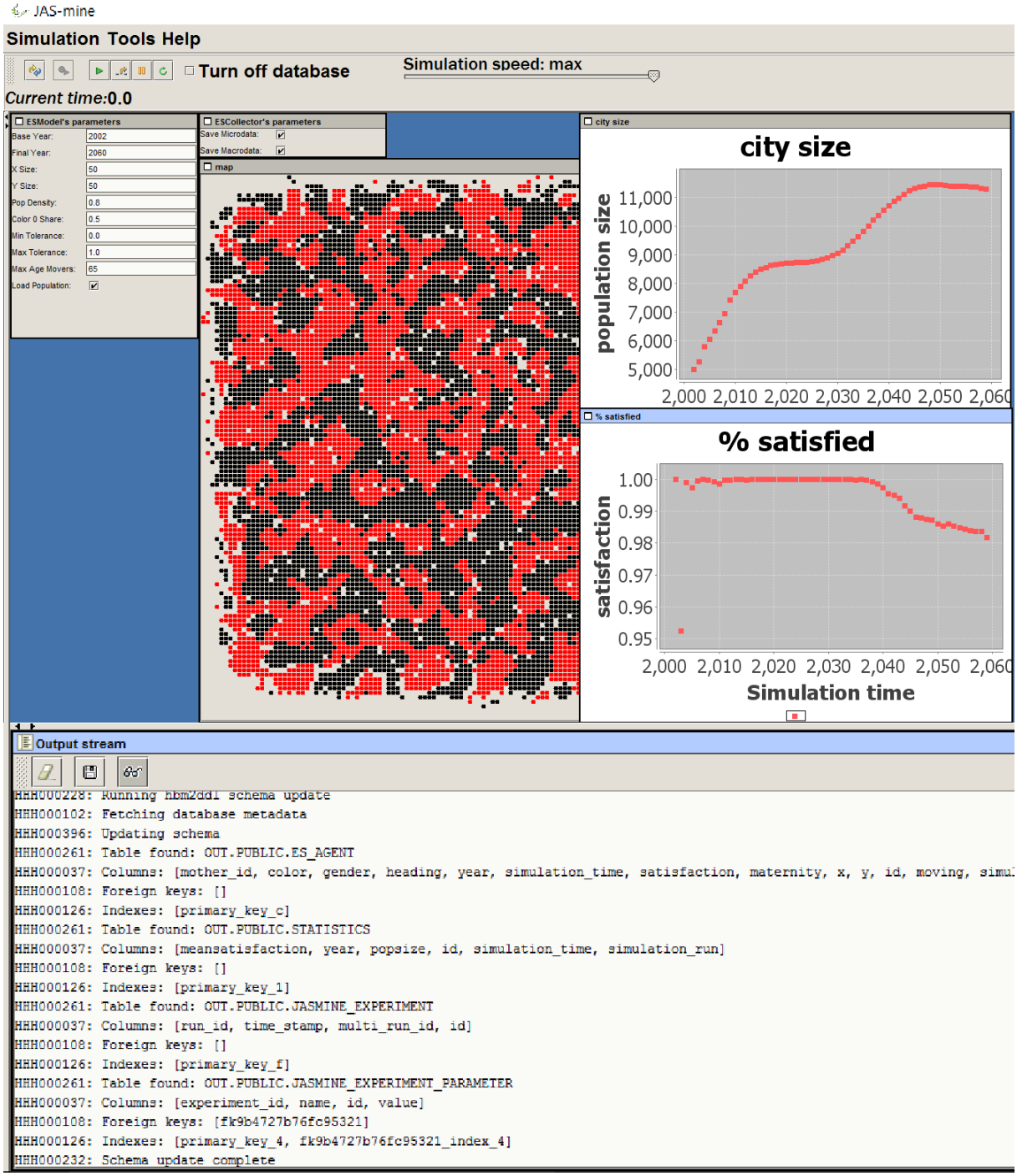
5 EXECUTION MODES AND GRAPHICAL USER INTERFACE

JAS-mine supports three different types of execution mode:- *interactive* mode, *batch* mode and *multi-run* mode. We discuss these modes in detail, and also provide a description of the main steps that occur 'under the bonnet' when a JAS-mine simulation is initiated.

5.1 Interactive mode and Graphical User Interface

The most common mode for prototyping a JAS-mine project, developing an intuition about how the underlying model works and demonstrating it to an audience is the interactive mode. This features a graphical user interface such as that illustrated in Figure 8, which allows users to inspect a number of the simulation's output quantities in real-time and assess the impact of a change in model parameters.[37] The GUI is built using the professional quality JFreeChart open-source Java libraries and can be pre-configured by a model developer to facilitate the use of JAS-mine simulation models by non-technical users such as policy makers, who can easily set parameter values and launch a simulation with just a few clicks of the mouse, and observe the resulting graphical and textual output on display.[38]

Figure 8: Interactive Mode



Screenshot of the JAS-mine graphical user interface showing output from the Extended Schelling demonstration model, available at *www.jas-mine.net/demo/extended-schelling*.

The GUI contains a control panel at the top that allows the user to pause the simulation, execute individual events step-by-step, and reduce the speed of the simulation (if, for example, the user deems it useful when observing the evolution of the model output). Below the control panel are parameter boxes listing *GUI parameters* and their default values, with a box for each Model, Collector and Observer featuring *GUI parameters*. The user can adjust parameters before executing the simulation, and there is also the possibility of updating the parameter values during the simulation. Below the parameters box, is the place where graphical output can be displayed. There is a rich graphical library allowing extensive visualization options in JAS-mine. As can be seen in Figure 8, graphics include time series plots and geographic maps. This screenshot of the JAS-mine GUI is from the Extended Schelling model, which combines the well known segregation model by Thomas Schelling (1969) that is illustrative of the AB approach, with demographic features of birth and death processes that are typical of dynamic microsimulations. The model can be downloaded from the JAS-mine website.[39] Below the charts is the output stream where textual information can be displayed, such as a model's output, running time and any error messages, in addition to information about any associated relational database.

The interactive mode is launched by default when executing the *Start* class of a standard JAS-mine project (as created using the JAS-mine Plugin for Eclipse IDE tool that was discussed in Subsection 4.6).

5.2 Batch mode

On the other hand, if the user desires to run a simulation model in the shortest possible time, JAS-mine can run in batch mode where the GUI and other unnecessary parts of the project can be switched off (e.g. the project's *Observer* class) in order to optimise speed of execution. This is possible due to the modular nature of JAS-mine code, embodying the JAS-mine philosophy of keeping conceptually distinct components of the project separate. The project can be run on High Performance Computing (HPC) clusters, offering the potential to run simulations that require much greater memory and processing power. In addition, Java has tools for parallelisation, concurrency and multithreading, enabling simulations to run across multiple cores if the user so desires.

5.3 Multi-run mode

Finally, the multi-run mode can be executed using the project's *MultiRun* class (as created using the JAS-mine Plugin for Eclipse IDE tool that was discussed in Subsection 4.6), that calls the JAS-mine simulation engine's multi-run tools to handle the sequential execution of simulations. This may be utilised to estimate the stochastic error of the simulations, facilitate 'design of experiments' (DOE) analysis and the optimisation of simulation output quantities.[40] In addition, parameter uncertainty analysis can be undertaken using the multi-run mode and JAS-mine's regression utility tools to bootstrap regression coefficients (see discussion in Section 4.3). The user should note that JAS-mine's input/output communication handling in multi-run mode is such that all data is exported into a single output relational database, indexed by the simulation time and run number. This allows the user to easily analyse the variation of output across simulation runs (and possibly over a variety of parameter domains) using their favourite statistical software (e.g. R, Stata, etc.).

5.4 Main Steps in a JAS-mine Simulation

A JAS-mine simulation begins when the main method of the Start class (found in a JAS-mine project's Experiment package) is executed.[41] This invokes an instance of the JAS-mine simulation engine, sets up the experiment builder, and creates the JAS-mine GUI if desired. If the JAS-mine GUI is indeed launched, the user can adjust any GUI parameter settings for the simulation run and then click on buttons in the GUI control panel to build the simulation experiment and start its execution. Otherwise, the main method in the Start class can call the simulation engine to start the simulation using the engine's startSimulation method (this is useful when running the simulation without the JAS-mine GUI in batch mode, for example on a high performance computer).

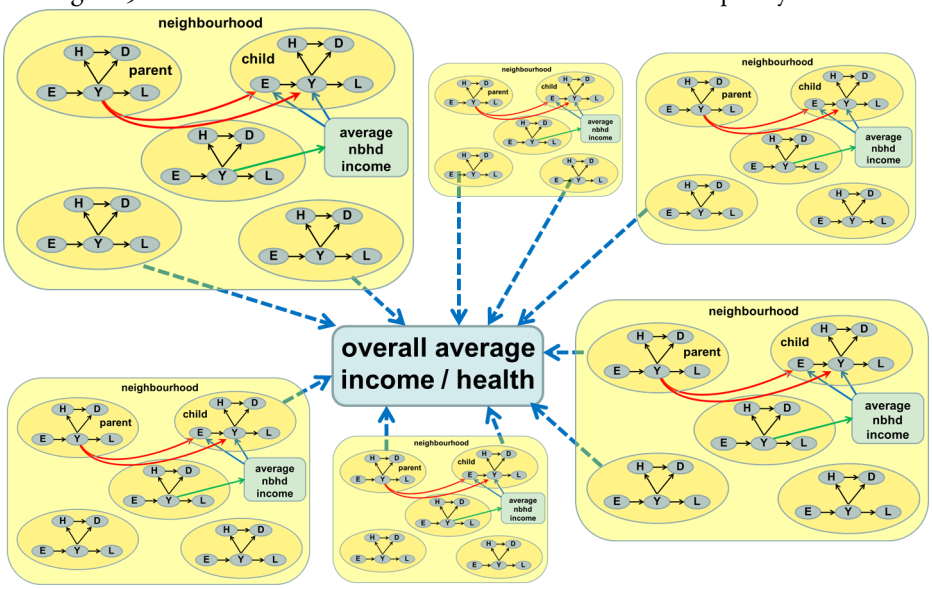
During the build process, the Start class' buildExperiment method is executed. The buildExperiment method constructs the *manager* classes (introduced in Subsection 3.1) required for the experiment and adds them to the simulation engine; this must include a Model class (found in the project's Model package) and can include Collector and Observer classes, and even additional Model classes if they exist.

When the manager classes are constructed, the buildObjects and buildSchedule methods in each class are called. The buildObjects method creates all the internal objects that are required for the class; the buildObjects method in a Model class typically creates a collection of agents to be simulated (with the blueprint for building an agent held in a separate Agent class), the buildObjects method in a Collector class may construct statistical objects (potentially from JAS-mine's statistical package in order to collect data from the agents, while a buildObjects method in an Observer class usually builds the graphical objects (such as those available in JAS-mine's GUI packages) required to display aspects of the state of the simulation that the user wishes to inspect.

The buildSchedule method in each manager class adds the class' events to the simulation's list of events.[42] The buildSchedule method in the Model class typically specifies events relating to repeated processes that evolve the population of agents over time, and can also include an event that terminates the simulation at a specified time in the future. The buildSchedule method in the Collector class usually contains events in which the model data is sampled and possibly exported to a comma-separated values (csv) file or the output database; the buildSchedule method also sets the frequency or times during the simulation at which these events occur. The buildSchedule method in the Observer class determines the frequency at which the graphical output of the simulation is updated in the GUI. This gives complete freedom to the modeller to specify how often to sample the model's data and how frequently to update any graphical output; the modeller may, for example, cut the time it takes a simulation to complete by reducing the frequency in which data is exported to output files or displayed in the GUI.

After the simulation run has started, the run ends when the simulation engine fires a termination event – such an event could have been scheduled for a specified simulation time, or could be contingent on some other events that might occur, such as the case when the population of agents become extinct so that there are no more agents to evolve. For an experiment involving only a single simulation run, the simulation is over and if the GUI was used, the graphical output will remain displayed on screen until the user closes the GUI shell window. For a multi-run experiment, the next simulation run (usually with a new set of parameters) will be built and executed and the cycle will continue until all the required simulation runs have terminated.

Figure 9: Interaction structure in the Theoretical Health Inequality Model



The interaction between states (attributes) of individual agents and across a hierarchy of aggregation levels in the Theoretical Health Inequality Model. For each agent, E is the education level, Y is the income level, H is the health index, D is the mortality probability and L is the geographic location. Source: Wolfson et al. (2016).

6 APPLICATIONS AND PERFORMANCE

We demonstrate the performance of JAS-mine by implementing a rich model, and assess how the time to execute the model is affected by the persistence of data to the output database or csv files.[43]

The Theoretical Health Inequality Model (THIM) was developed by Wolfson et al. (Wolfson et al., 2016) to understand why cities in the US and UK with higher income inequality have lower health-adjusted life expectancy, while Australian, Canada and Swedish cities do not. The model is motivated by the theory that low income households living near high income households tend to benefit from better infrastructure and amenities such as hospitals, which may be denied to them if living in a city partitioned into rich and poor neighbourhoods (the "gates and ghettos" case). The model specifically tries to demonstrate this relationship by varying the heterogeneity within each neighbourhood and across neighbourhoods.

THIM is a computationally heavy model with lots of interactions, however agents also undergo demographic transitions modelled by stochastic processes represented by regressions, so it is ideal in representing a hybrid AB and MS model. The model recreates stylized individual-level relationships among health status, education, income, mortality rates and neighbourhood mobility. There are multiple levels featured in THIM, from individuals and parent-child dyads, to neighbourhoods and cities. The interactions between the states of individuals and across a hierarchy of aggregation levels are represented in Figure 9. These levels allow to capture the roles of parental transmission of socio-economic status and health advantage to children, the impact of average neighbourhood income on school, and overall city-wide patterns of inequality and mortality.

THIM features a mix of regular events; each agent updates its status once a year on its birthday, and the system-wide statistics against which an agent measures itself are updated at the start of a new year. There are also irregular events and dynamic scheduling; the time at which each agent gives birth and dies is drawn probabilistically

from continuous distributions during the simulation. This means that we cannot know *a priori* the timing of events at the start of the simulation, and therefore cannot use JAS-mine's scheduling tools to reduce the number of events that need to be scheduled by scheduling events to apply to the whole population of agents, as we did in the Demo07 demonstration model (Richiardi & Richardson, 2016). In this case, the model's event schedule scales with the number of agents and not just the number of processes. Simulating a country on a one-to-one scale would mean potentially having to schedule hundreds of millions of events during run-time.

The priority queue behind JAS-mine's event schedule keeps the access and insertion of events in the event schedule computationally efficient, with access to the earliest event achieved in constant time, whilst insertion is performed in logarithmic time O(log N) in the worst case. This means that a model simulating the United States on a one-to-one scale containing 300 million agents would only need to check up to around thirty events of the schedule to find the correct place to insert a new event.[44]

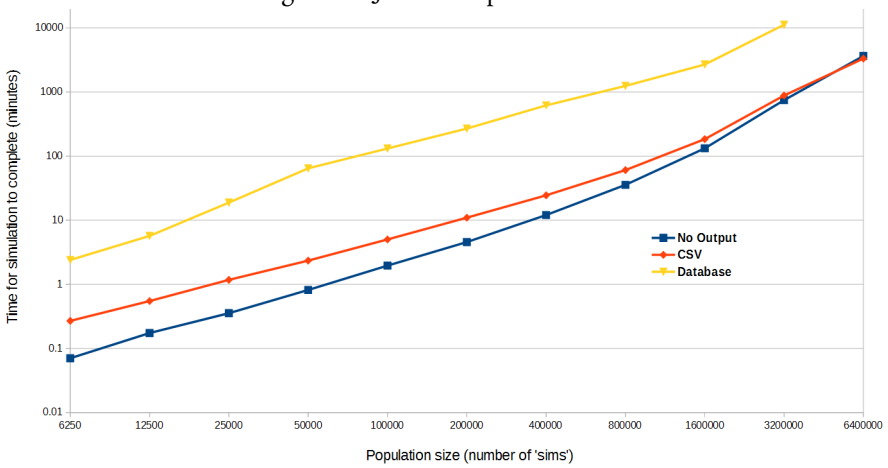
We performed the simulations using the University of Oxford's Advanced Research Computing cluster ARCUS (Phase B), which features Intel E5-2640v3 Haswell processors and up to 128GB of random access memory.[45] In order to assess the impact that the persistence of data has on performance in JAS-mine, simulations for a wide range of population sizes were run for three different data exporting modes:- persistence to a relational database, the export of data to csv files, and finally the benchmark setting with no recording of data. The data that is exported by the simulation is the whole state of the population at the start of every simulated 'year' for five-hundred years.

The time taken for the simulations to complete under the different data export modes can be seen in Figure 10, for population sizes doubling from 6250 agents up to 6.4 million agents (the result for 6.4 million agents with data exported to the relational database did not complete within a 10 day time-frame). Note that the time taken for the simulations to complete doesn't just depend on the platform (which will be as fast as implementing the model in pure Java), but also on the structure of the model, how it is implemented and even the computer architecture used.[46] Indeed, how the time scales with population size – in this case, the time taken to complete appears to scale approximately as a power law over the range of investigation – is a property of the structure of the model, influenced by the nature of interaction between agents. The figure, which shows a convergence between the times taken for simulations with no output and data exported to csv files as the population size increases, demonstrates that the time taken for data to be exported represents a diminishing proportion of the overall simulation time.

It is important to assess the difference between the modes of data export, with respect to the benchmark of no data output. The export mode persisting data to the output database illustrates the additional overhead involved in running simulations with the underlying machinery of object-relational mapping (ORM). Indeed, the figure shows that the additional time costs of exporting the output to csv files is negligible when compared to the additional time it takes to persist the output of the THIM to the relational database. We leave it to the user to decide the best data export option for his or her needs, which will depend on the nature and scale of the model they develop, along with how the user intends to perform data analysis on the simulation output (i.e. whether the benefits of storing data in a relational database justify the additional time costs).

For completeness, it should also be noted that the relational database files are about 20% larger than the csv files, with files ranging from around 400MB for THIM simulations with an initial population size of 6250, up to

Figure 10: JAS-mine performance.



The time taken (in minutes) for simulations of THIM implemented in JAS-mine to complete for a variety of population sizes. The three lines correspond to different data export options, with 'Database' referring to persistence to the output relational database, 'CSV' referring to the export of data to separate comma-separated values files, and 'No Output' referring to the benchmark case where no data is exported. Simulations were executed on the ARCUS-B cluster of the University of Oxford's Advanced Research Computing facility.

400GB for an initial population size of 6.4 million agents.

7 CONCLUSIONS

In this paper, we have introduced the JAS-mine platform, a Java-based toolkit for discrete-event simulations specifically designed to aid development of agent-based and dynamic microsimulation models, anticipating a convergence between the two fields. As discussed in Richiardi & Sonnessa (2013), the platform can be assessed both with respect to what it is, and what it is not. First, JAS-mine is not a tool to speed up simulation execution – its execution speed will be the speed of Java; rather, its goal is to speed up model *development*, facilitate model *documentation*, and foster model *testing* and *sharing*. The rationale behind this choice lies in the observation that computer power is always increasing, while developers' time is not. Also, large-scale simulation projects are generally beyond the reach of a single researcher. Even when they remain under the control of a restricted group of people, simulation projects generally require a prolonged effort, often on a stop-start basis. The possibility of building on work done in the past by the same authors or by other researchers is crucial. Simulation modelling needs cooperative development, and the choice of an entirely open-source tool should be evaluated in this light.

In the trade-off between efficiency and transparency, we deliberately opt for the latter. However, JAS-mine does not force the user to adopt predefined solutions to the problems faced in simulation modelling. By offering a set of libraries that extend the capability of the standard Java classes, JAS-mine leaves entirely open the possibility of using other libraries and tools, either as an alternative or on top of the JAS-mine toolkit. This is similar to other platforms such as MASON and RePast, which are also Java-based and open-source. However, these simulation toolkits leave input/output communication somewhat in the backyard, and are therefore ill-suited for microsimulation modelling.

From a modelling perspective, the main value added by JAS-mine is the inclusion of specific libraries for regression modelling, alignment and uncertainty analysis. From a computer science perspective, the main value added lies in the integration of an object-oriented simulation platform with a relational database, through the use of

object-relational mapping. Clearly, our approach is an overkill for small-scale agent-based models. Toy models designed to provide insight into the relevant mechanisms of social interaction do not generally need relational archives for input and output. However, the use of large-scale agent-based macro models is becoming increasingly popular as an alternative to the standard DSGE approach in macroeconomics. At the same time, our methodological proposal of strictly separating (*i*) model specification (the agents and their environment), (*ii*) micro and macro algorithms (the econometric formulas used for predicting outcomes at an individual level and the specific methods used for alignment and matching), (*iii*) data collection and analysis, could also be useful for dynamic microsimulation modelling. This separation is possible thanks to a strict adherence to an object-oriented approach and a detailed package structure. The price to pay, for instance with respect to LIAM2 or Modgen (which feature their own idiosyncratic syntaxes based respectively on Python and C++), is a slightly more involved syntax. The benefits of JAS-mine, however, include the possibility to extend the platform in endless directions due to its open-source architecture, the readability that comes with an object-oriented approach especially when the models scale up, and the power and flexibility given by the possibility of storing the underlying data in a relational database or in comma-separated values (csv) files. As with most things, diversity is a strength, and in this light we hope JAS-mine will be welcomed in the agent-based and dynamic microsimulation communities.

ACKNOWLEDGEMENTS

The authors acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. http://dx.doi.org/10.5281/zenodo.22558. For this research, Matteo Richiardi benefited from support by a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme.

REFERENCES

Baekgaard, H. (2002). *Micro-macro linkage and the alignment of transition processes: some issues, techniques and examples* (Tech. Rep. No. 25). National Centre for Social and Economic Modelling (NATSEM).

Bilcke, J., Beutels, P., Brisson, M., & Jit, M. (2011). Accounting for methodological, structural, and parameter uncertainty in decision-analytic models: A practical guide. *Medical Decision Making*, *31*(4), 675–692.

Creedy, J., Kalb, G., & Kew, H. (2007). Confidence intervals for policy reforms in behavioural tax microsimulation modelling. *Bulletin of Economic Research*, *59*(1), 37–65.

De Menten, G., Dekkers, G., Bryon, G., Liègeois, P., & O'Donoghue, C. (2014). Liam2: a new open source development tool for discrete-time dynamic microsimulation models. *Journal of Artificial Societies and Social Simulation*, *17*(3), art. 9.

Gilbert, N., & Terna, P. (2000). How to build and use agent-based models in social science. *Mind & Society*, *1*(1), 57–72.

Keller, A., Agarwal, S., & Jensen, R. (1993). Enabling the integration of object applications with relational databases. In *Proc. of acm-sigmod.*

Klevmarken, A. (2002). Statistical inference in micro-simulation models: incorporating external information. *Mathematics and Computers in Simulation*, *59*, 255–265.

Leombruni, R., & Richiardi, M. (2006). Laborsim: An agent-based microsimulation of labour supply. an application to italy. *Computational Economics*, *27*(1), 63–88.

Li, J., & O'Donoghue, C. (2013). A survey of dynamic microsimulation models: uses, model structure and methodology. *International Journal of Microsimulation*, *6*, 3-55.

Li, J., & O'Donoghue, C. (2014). Evaluating binary alignment methods in microsimulation models. *Journal of Artificial Societies and Social Simulation*, *17*(1), art. 15.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, *81*(7), 517–527.

Luna, F., & Stefansson, B. (2000). *Economic simulations in swarm: Agent-based modelling and object oriented programming*. Kluwer.

Mannion, O., Lay-Yee, R., Wrapson, W., Davis, P., & Pearson, J. (2012). Jamsim: a microsimulation modelling policy tool. *Journal of Artificial Societies and Social Simulation*, *15*(1), art. 8.

Minar, N., Burkhart, R., Langton, C., & Askenazi, M. (1996). *The swarm simulation system: A toolkit for building multi-agent simulations* (Working Paper No. 96-06-042).

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., & Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, *1*(1), 1–26. Retrieved from `http://dx.doi.org/10.1186/2194-3206-1-3` doi: 10.1186/2194-3206-1-3

OpenM++. (2013). Openm++: open source microsimulation platform. http://ompp.sourceforge.net [Computer software manual]. Retrieved 21st April 2016, from `http://ompp.sourceforge.net`

Richardson, R. E., Pacelli, L., Poggi, A., & Richiardi, M. (2016). *Female labour force projections using microsimulation for six eu countries.* (Tech. Rep.). Institute for New Economic Thinking at the Oxford Martin School.

Richardson, R. E., & Richiardi, M. (2016). *Understanding low labour force participation: Policy evaluation using microsimulation.* (Tech. Rep.). Institute for New Economic Thinking at the Oxford Martin School.

Richiardi, M. (2013). The missing link: Ab models and dynamic microsimulation. In S. Leitner & F. Wall (Eds.), *Artificial economics and self organization* (Vol. 669). Springer.

Richiardi, M., & Poggi, A. (2014). Imputing individual effects in dynamic microsimulation models. an application to household formation and labor market participation in italy. *International Journal of Microsimulation*, *7*(2), 3–39.

Richiardi, M., & Richardson, R. E. (2016). Agent-based computational demography and microsimulation using jas-mine. In A. Grow & J. van Bavel (Eds.), *Agent-based modelling in population studies: Concepts, methods and applications.* Springer. doi: 10.1007/978-3-319-32283-4

Richiardi, M., & Sonnessa, M. (2013). *Jas 2: A new java platform for agent-based and microsimulation modeling* (Working Paper No. 134/2013). LABORatorio Revelli. Retrieved from `http://www.laboratoriorevelli.it/_pdf/wp134.pdf`

Schelling, T. (1969). Models of segregation. *American Economic Review*, *59*, 488–493.

Sonnessa, M. (2004). Jas: Java agent-based simulation library. an open framework for algorithm-intensive simulations. In R. Leombruni & M. Richiardi (Eds.), *Industry and labor dynamics: The agent-based computational economics approach.* World Scientific Press.

Statistics Canada. (2009). Modgen version 10.1.0 developer's guide. http://www.statcan.gc.ca/sites/default/files/dev-guide-eng.pdf. [Computer software manual]. Retrieved 21st April 2016, from `http://www.statcan.gc.ca/sites/default/files/dev-guide-eng.pdf`

Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo/. [Computer software manual]. Evanston, IL.. Retrieved 21st April 2016, from `http://ccl.northwestern.edu/netlogo/`

Wolfson, M., Gribble, S., & Beall, R. (2016). Exploring contingent inequalities - building the theoretical health inequality model. In A. Grow & J. van Bavel (Eds.), *Agent-based modelling in population studies: Concepts, methods and applications.* Springer.

NOTES

[1] A number of third-party solutions for parallelising Java code are readily available.

[2] See www.jas-mine.net.

[3] Comparisons of different microsimulation modelling tools can be found in (Li & O'Donoghue, 2013) and (De Menten et al., 2014).

[4] LIAM2 models are written in a custom language derived from Python, while Modgen models are written in a proprietary language similar to C++.

[5] Java and Scala are interoperable, meaning that Java libraries may be used directly in Scala code and vice versa.

[6] RePast was originally written in Java (RePast J), and is also available for the Microsoft.Net framework (Repast.Net).

[7] See http://www.antoinegodin.eu/pksfc.

[8] Modelling interaction between individuals is definitely out-of-bounds in Stata.

[9] For a comparison of agent-based platforms, see https://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software.

[10] When computation time becomes an issue, specific features of JAS-mine can be switched off, to revert to pure Java performance (see Sections 5 and 6).

[11] The source code is available on GitHub at https://github.com/jasmineRepo/JAS-mine-core and https://github.com/jasmineRepo/JAS-mine-gui.

[12] See www.jas-mine.net/home/documentation.

[13] See the Job applications demo model at www.jas-mine.net/demo/job-applications.

[14] See www.jas-mine.net/home/documentation/cookbook/queries for more details.

[15] JAS-mine uses the Hibernate H2 database, see Subsection 3.3.2.

[16] See www.h2database.com/html/main.html and www.hibernate.org.

[17] See www.jas-mine.net/home/documentation/focus/object-relational-mapping for more details.

[18] See www.jas-mine.net/home/documentation/cookbook/Output-to-CSV-files-and-the-database.

[19] Note that Oracle recommends using a Windows release later than XP to run their implementation of Java 7.

[20] See the 'Limits and Limitations' section at www.h2database.com/html/advanced.html.

[21] See www.jas-mine.net/home/documentation/cookbook/the-model-and-the-schedule.

[22] See http://dst.lbl.gov/ACSSoftware/colt/index.html and http://commons.apache.org/proper/commons-math.

[23] See www.jas-mine.net/home/documentation/tutorials/how-to-use-the-jasmine-statistical-package.

[24] See www.jas-mine.net/demo, with source code found at https://github.com/jasmineRepo.

[25] See www.jas-mine.net/home/documentation/focus/uncertainty-analysis.

[26] See Subsection 5.3 for discussion of JAS-mine's multi-run capabilities.

[27] See www.jas-mine.net/home/documentation/cookbook/alignment.

[28] Resampling Alignment is used in the Labour Force Participation demonstration model (Richardson et al., 2016; Richardson & Richiardi, 2016), see www.jas-mine.net/demo and in particular the implementation in the source code at https://github.com/jasmineRepo/LabourForceParticipation.

[29] A discussion of how to implement SBD Alignment can be found in Richiardi & Richardson (2016) and at www.jas-mine.net/demo/demo07/personsmodel.

[30] See www.jas-mine.net/demo/demo07/personsmodel for implementation details.

[31] See www.jas-mine.net/home/documentation/cookbook/matching.

[32] See https://eclipse.org.

[33] See https://marketplace.eclipse.org/content/jas-mine-plugin-eclipse-ide.

[34] See https://maven.apache.org.

[35] See www.jas-mine.net/demo/labour-force-participation.

[36] See www.jas-mine.net/demo/thim.

[37] The JAS-mine GUI is described in detail at www.jas-mine.net/home/documentation/cookbook/gui.

[38] See http://www.jfree.org/jfreechart.

[39] See www.jas-mine.net/demo/extended-schelling.

[40] See www.jas-mine.net/home/documentation/tutorials/run-a-simulation-many-times.

[41] For a detailed description of the file structure of a JAS-mine project, see http://www.jas-mine.net/home/documentation/tutorials/the-structure-of-a-jasmine-project.

[42] The simulation's list of events is held in a singleton object.

[43] See www.jas-mine.net/demo/thim for links to the source code.

[44] This assumes the priority queue implementation uses a binary heap structure.

[45] See www.arc.ox.ac.uk/content/services.

[46] We find that simulations persisting data to an output relational database complete in a shorter time on a personal computer than on ARCUS-B; running on a personal computer, the time to complete for database persistence is around six times longer than exporting to csv files, versus eighteen times longer when running on ARCUS-B. The reverse, however, is true for exporting to csv files and not exporting any data, which complete in a shorter time on ARCUS-B. The main reason we choose to run simulations on a high performance computer is to assess how the model scales with larger population sizes that, due to the large memory and storage space requirements, are not possible to execute on a typical personal computer at the time of writing. Note that the model has not been redesigned to take advantage of the parallelization possibilities on ARCUS-B, which would further increase the speed of execution.