SoftwareX 12 (2020) 100555

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Dymium: A modular microsimulation modelling framework for integrated urban modelling

Amarin Siripanich, Taha Hossein Rashidi [*]

UNSW Sydney, Research Centre for Integrated Transport Innovation (RCITI), School of Civil and Environmental Engineering, NSW, 2052, Australia

ARTICLE INFO

Article history:
Received 20 January 2020
Received in revised form 18 June 2020
Accepted 18 June 2020

Keywords:
Microsimulation
Urban modelling
Life-course
R package

ABSTRACT

Dymium is a microsimulation framework written in R for integrated urban modelling. It aims to solve technical challenges faced by microsimulation modellers and make the approach more accessible to a broad range of users. With numerous packages that are available for free in R, including Dymium, all the stages – data preparation, microsimulation, calibration, and model validation – in building a microsimulation model can be done effectively under one programming environment. Although we developed the framework for modelling urban systems, i.e., demography, firmography and transport modelling, it has all necessary functionalities that can be readily adopted for other contexts.

Code metadata

Current code version	*v* 0.1.9
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_8
Legal Code License	GPLv3
Code versioning system used	git
Software code languages, tools, and services used	R (≥3.5.0)
Compilation requirements, operating environments & dependencies	*Linux, OS X, Microsoft Windows. Runs within the R software environment.*
If available Link to developer documentation/manual	https://core.dymium.org
Support email for questions	amarin@dymium.org

1. Motivation and significance

An analytical tool developed on a microsimulation foundation enables planners and decision makers to evaluate the impacts of new policies or hypothetical changes in a virtual world [1]. This analytical approach offers a highly disaggregated view on changes across time, places, decision makers, and markets. It allows behavioural realistic models to be applied at the individual level which predict the actions and the attributes of heterogeneous agents and their evolution over time while taking the effects of path dependence into account.

Microsimulation has been widely applied across multiple areas from demography [2,3], public policy [4], pension [5], taxation [6], economics [7,8], public health [9–11], epidemiology [12, 13], transport [14–16] and more. Thus, it is not hard to see why this versatile analytical approach has been appreciated across a wide range of domains. In the context of urban systems, it has been applied to explore a multitude of questions, for example residential relocation [17,18], firm relocation [19,20], travel demand forecasting [15], urban energy consumption [21], interaction between transport and land use [22]. Also, it is not foreign to government agencies around the world [23,24].

In the most basic form, an integrated urban model (IUM) must consist of modules that represent population, employment, land use and transportation. IUMs use feedback loops between different sub-systems to represent complex interrelationships between different parts of the system, such as the interaction between the transportation system and the land use layer [25].

Some major weaknesses of microsimulation are high costs of model implementation and maintenance [26], high computational cost [27], long calculation time [17], data hunger [28], and along with other problems inherent large-scale model as argued by Lee [29].

Dymium aims to provide the fundamental building blocks for constructing a microsimulation model of urban systems. It should

* Corresponding author.
E-mail addresses: a.siripanich@unsw.edu.au (A. Siripanich), rashidi@unsw.edu.au (T.H. Rashidi).

https://doi.org/10.1016/j.softx.2020.100555

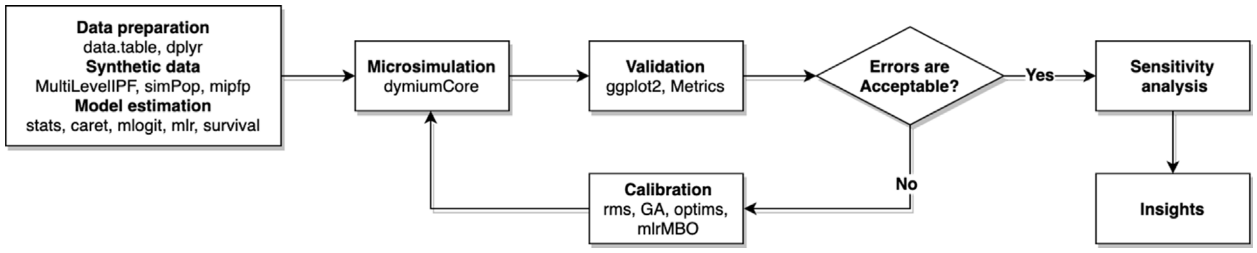

Fig. 1. Simplified flowchart of the stages in microsimulation modelling (in bold) and R packages for those stages.

be clearly stated that Dymium does not offer methodological contributions to the microsimulation approach. Instead, our goal is to provide a microsimulation toolbox that helps streamline microsimulation workflows by using an interpreted programming language that is familiar to many analysts. In a significant way, many technical challenges have been hindering the rate of the scientific discoveries being made in the field of microsimulation [30], this is also true for integrated urban modelling frameworks that use a microsimulation approach [26]. Hence, we believe contribute to the field by developing a rigorous and less costly microsimulation platform.

Solving the complicated workflow of microsimulation – R, an interpreted high-level programming language, is highly effective for data analysis, statistical modelling and graphical display. Many crucial steps – from data preparation, simulation, calibration to validation – that are required in building a microsimulation model can be done effectively and effortlessly using freely available R packages, see Fig. 1. Having all the analysis steps under one programming environment is highly beneficial in terms of reproducibility and maintainability, which places a huge mental burden on the modeller's mind and those who want to replicate the work. The R notebook, equivalence of Python's Jupyter notebook, can help the user to iterate in all aspects of their microsimulation workflow efficiently. Moreover, there is no need to manually transfer the parameters of models estimated in other statistical software to be used in their microsimulation model since the user can utilise modelling packages in R to estimate a model. On top of that, Dymium knows how to use many R's model objects for simulation, making the workflow even simpler even if the user needs to re-estimate their models. In this aspect, UrbanSim are much more limited to only statistical models that they implemented, while ILUTE and SILO (open-source urban land use models) required model parameters to be hard coded into their source code.

Designed for a board range of users – Dymium is developed with the needs of different levels of users in mind. It is fairly simple to get started for novices by the plug-and-play style using the available modules in the 'dymiumModules' repository. At the same time, it allows experienced users the freedom to develop new modules and modify or extend existing ones. Integrating modelling frameworks from other programming environments to the R environment can be done with ease using interface packages such as 'Rcpp' for C++, 'rJava' for Java, 'rticulate' for Python, and 'rJulia' for Julia. Being written in R should make the framework more accessible to a broader audience. On top of that, a graphical user interface for Dymium is being developed, which will enable high-level users (e.g. decision makers) to interact with Dymium models.

Open source – Dymium's source code is freely distributed and its will always be free to use for any purpose. Many microsimulation models of urban systems, with very few exceptions such as UrbanSim, that have been published or used in practice are often unavailable for public scrutiny. This means that most researchers interested in applying such techniques have to reinvent 'the wheel'. However, it cannot be denied that having such tools available at hand within a research group can create a significant leverage over other research groups which likely leads to more successful publications and reputation in the field.

To the authors' knowledge, Dymium is the first urban microsimulation package in R. That being said, it should be pointed out that there are very few packages such as 'MicMac' and 'simario' that offer microsimulation modelling capabilities but lack many features to support modularity, extensibility, shareability, and complexity for a microsimulation model's implementation like Dymium is capable of.

2. Software description

Dymium provide a set of R packages that aims to facilitate microsimulation model creation and simulation that are fast, modular, customisable and scalable to the requirements of each study. The full documentation can be viewed in the package or on the website. As of now, the packages as shown in Fig. 2 are distributed via GitHub[1] and can be easily installed. A comprehensive set of domain level modules is available at the dymiumModules repository.[2] This allows new users to get started quickly by using the existing modules and modify them to better suit their use cases. The remaining of this section aims to explain our design considerations and provides an overview of the main components and functionalities of Dymium.

2.1. Design considerations

To focus on providing functions for microsimulation modelling tasks, Dymium relies on other R packages for different purposes, from basic data operations to parallelisation. Although, Dymium does not provide built-in plotting functions at the moment, but it provides functions that can easily export simulation data so that users can use their favourite visualisation tools, such as ggplot2, to make plots. It is worth noting that, the 'R6' [31] and the 'data.table' [32] packages are the real backbone of Dymium.

We utilise the 'R6' class system, an implementation of encapsulated object-oriented programming (OOP) for R, for all the classes in Dymium. Since 'R6' objects have reference semantics and are mutable, we can construct a pipeline of microsimulation event functions and pass an object which acts as a container, storing pointers to entity objects and models in the memory, to sequentially simulate each event, as illustrated in Fig. 3. Hence, the entire microsimulation model can be modularly constructed, resulting in a code that is easy to understand at the high level.

Data.frame, like many objects in R, is immutable. The inner workings of a microsimulation model involve modifying data and adding/removing rows and variables. However, these basic data operations are very costly to perform on objects that are

1 https://github.com/dymium-org.
2 https://github.com/dymium-org/dymiumModules.

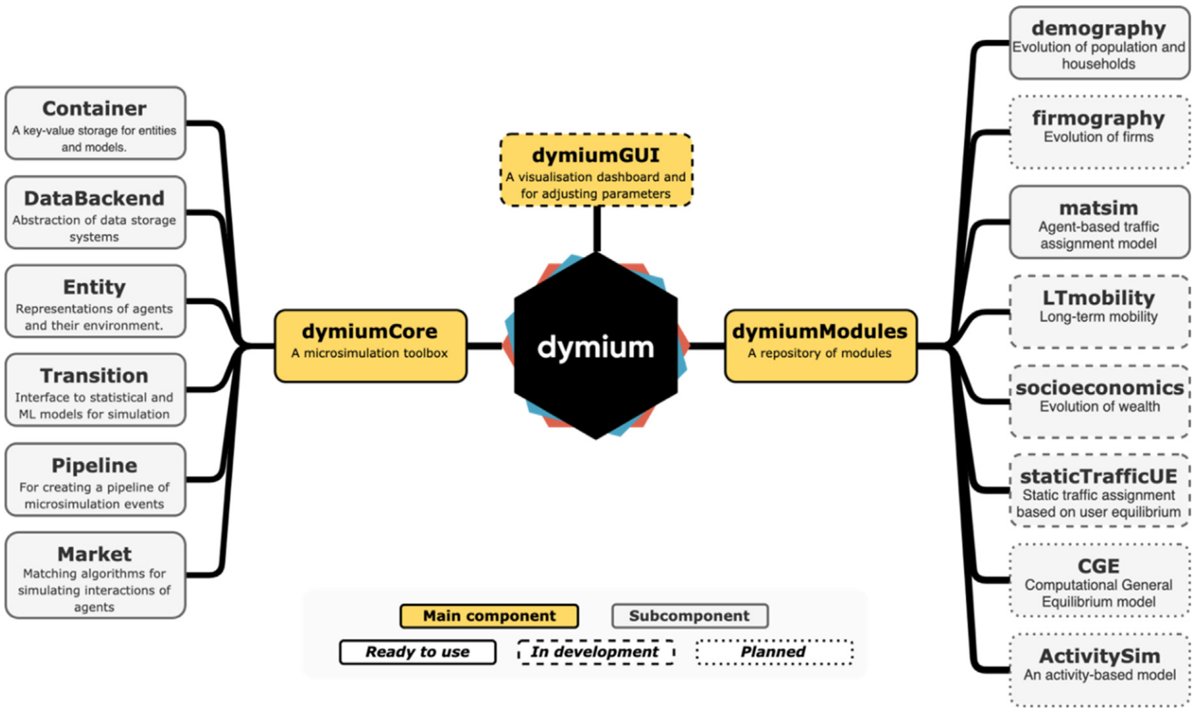
Container
A key-value storage for entities and models.
DataBackend
Abstraction of data storage systems
Entity
Representations of agents and their environment.
Transition
Interface to statistical and ML models for simulation
Pipeline
For creating a pipeline of microsimulation events
Market
Matching algorithms for simulating interactions of agents
dymiumCore
A microsimulation toolbox
dymiumGUI
A visualisation dashboard and for adjusting parameters
dymium
dymiumModules
A repository of modules
demography
Evolution of population and households
firmography
Evolution of firms
matsim
Agent-based traffic assignment model
LTmobility
Long-term mobility
socioeconomics
Evolution of wealth
staticTrafficUE
Static traffic assignment based on user equilibrium
CGE
Computational General Equilibrium model
ActivitySim
An activity-based model
Main component
Subcomponent
Ready to use
In development
Planned

Fig. 2. An overview of Dymium packages and modules, and the development plan.

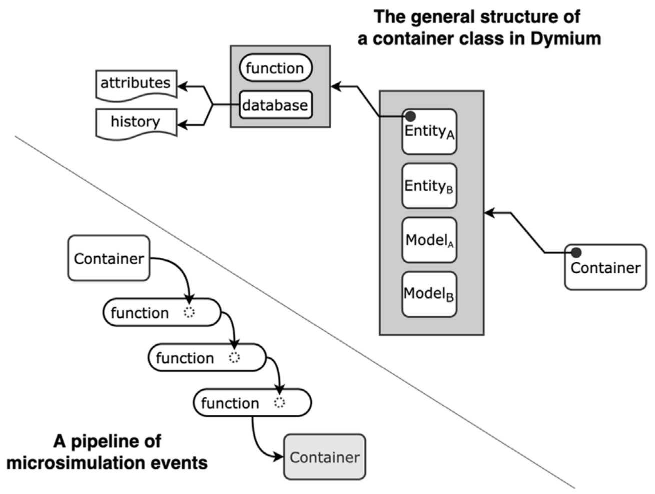

Fig. 3. A high-level implementation of a microsimulation model built with Dymium.

immutable such as data.frame, due to their copy-on-modify[3] behaviour. Hence, Dymium was designed to store all data that need to be frequently changed during simulation (e.g. attribute data of entities) as data.table objects. A 'data.table' object is essentially a high-performance version of base R's data.frame, where many common operations have been optimised using low-level parallelism in C++ and it can be modified in place. As a result, most operations inside Dymium models are automatically done in parallel and also memory efficient at the same time. All data are stored in memory during simulation to make sure the model runs as fast as possible.

Within an R session, connections to external databases can be made using packages such as DBI, ODBC, RSQLite, mongolite, RMySQL, and RPostgreSQL. These packages provide interfaces to work with various databases from R. Unstructured databases can be used as long as they can be converted into list columns inside a data.frame object. Efficiency-wise, we recommend against using list columns as most data operations are non-vectorisable, meaning they are generally slower than non-list columns.

In terms of scalability and distributed computing, all microsimulation models built with Dymium can be executed in sequential and parallel on a set of local machines, remote machines or a mixed of the two with ease, thanks to the APIs provided by the 'future' package [33].

2.2. Software architecture and functionalities

This subsection introduces the fundamental building blocks for creating a microsimulation model provided in the dymiumCore package, as shown in Fig. 2. Additionally, many helper functions and classes, although not fully discussed here, are also available to the user to facilitate their model development and execution such as the Market class which simulates interactions of agents (i.e. mate matching, firm-job matching, real-estate bidding), and helper functions to create a new module using a recommended structure to ensure reusability.

2.2.1. Entity and container

Economic agents (i.e. individuals and households), spatial boundaries, assets (i.e. buildings and vehicles), infrastructure (i.e. transportation networks) are entities in Dymium. An entity can be described by its attributes and methods. As shown in Fig. 4, an instance of the *Individual* class is meant to store individuals' attributes and has methods that help in identifying relations to other individual agents in the population, adding new individual agents, and so on.

When there are multiple Entity inheritances that are intrinsically linked to one another (i.e. one is a member of the other) such as the *Individual* class and the *Household* class, we may use a container class to store functions for accessing and modifying attributes of the related instances simultaneously. For example,

[3] https://adv-r.hadley.nz/names-values.html#copy-on-modify.

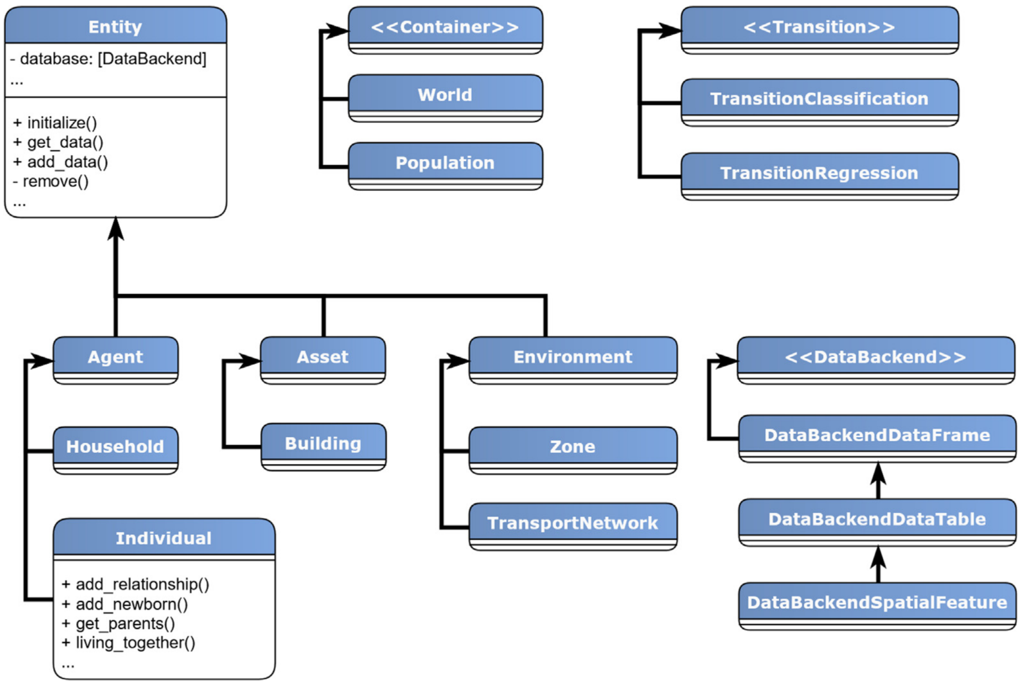

Fig. 4. A simplified UML class diagram of the main building blocks of Dymium.

Table 1
List of available and in development modules in the 'dymiumModules' repository.

Module name	Events	Status
Demography	1. Aging 2. Death 3. Birth 4. Marriage 5. Separation 6. Divorce 7. Cohabitation 8. Breakup 9. Leave parent home 10. Migration	Ready to use
Matsim	1. Run MATSim 2. Create demand	Ready to use (requires Java)
Firmography	1. Enter market 2. Exit market 3. Firm relocation 4. Location search 5. Growth and contraction	In development
Mobility	1. Residential relocation 2. Housing search 3. Change job 4. Job search	In development
Socioeconomics	1. Income 2. Education status 3. Labour force status 4. Driving licence 5. Occupation	In development

Population is a container class that when it is instantiated it will also instantiate an instance of *Individual* and *Household* that are linked. *Population* can modify the databases of individual and household agents, and every time either of the agent groups have been removed, it will also update the database of the other instance to maintain the validity between them (i.e. remove the individuals that belonged to the removed households).

The *DataBackend* classes enable different entity types to have a data structure that best represent them. For example, the Zone class and the Transportation Network class are spatial, which means they require geographical information to describe them. Hence, the best data structure for them is the *DataBackendSpatialFeature* class which helps to retain their geographical information necessary for making spatial visualisation and perform spatial operations. Other classes that do not require to be spatially represented may use *DataBackendDataTable* (using the data.table package as the data backend) for efficient data operations. Additionally, an instance of Entity class, as well as its inheritances, can have multiple databases associated with it and each database may use different *DataBackend* classes. For instance, individuals' daily travel activities should be stored separately from their main attributes (i.e. age, gender, marital status, etc.) to avoid creating a complex data structure that can potentially create inefficiency down the road.

Another important class that should be discussed is *World*. It is a container class that we use for storing entities, models and other simulation parameters. For this reason, event functions are designed to take, and also return, a World object as their first argument so that they can access the entities and models encapsulated within the World object.

2.2.2. Event, transition and module

Events in Dymium are asynchronous, meaning they do not all occur at the same time. They are essentially R functions that mutate the state of an instance of *World* or *Entity*'s sub-classes in discrete time. An event function may contain more than one transition that the targeted instance(s) of *Entity*'s inheritances require to undergo. Such as in a marriage event, where individuals first have to make marital choice, then those who decided to get married have to select a suitable mate, follow by the choice of household merging or formation.

TransitionClassification and *TransitionRegression* allow model objects to be used for simulation and alignment [34]. They currently support many models fitted using *caret*, *mlr* and *mlogit*. The caret package alone provides a consistent interface for model fitting and prediction to 238 different models[4] including many

[4] see the full list of the available models at https://topepo.github.io/caret/available-models.html.

Table 2
Summary of the events in the example microsimulation model.

Sequence	Event	Model type	Time resolution
1	Ageing	Deterministic	Yearly
2	Migration	Deterministic	
3	Birth	Logistic regression	
4	Death	Rate-based	
5	Marriage	Logistic regression	
6	Divorce	Logistic regression	
7	Cohabitation	Logistic regression	
8	Breakup	Logistic regression	
9	Leaving parental home	Logistic regression	
10	Assign daily travel activities	Statistical matching	Every five years
11	Traffic assignment	Agent-based simulation	

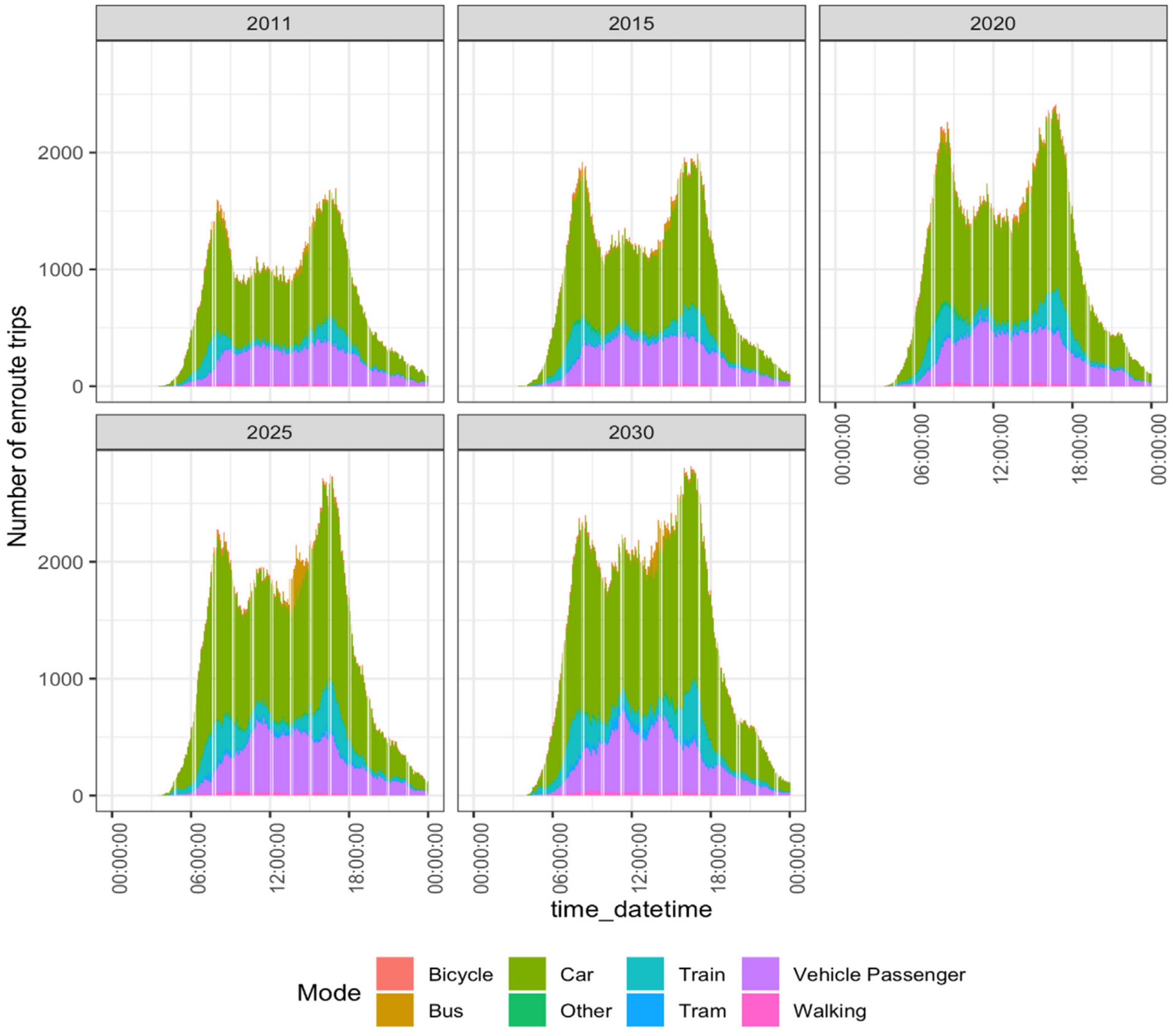

Fig. 5. Forecasts of the number of en route trips by modes and time of day on an average day.

well-known R packages for estimating econometric models to advanced machine learning models.

A very important feature of Dymium that needs to be highlighted is modularity. It is utmost important for a microsimulation model to be maintainable and flexible to changes without interrupting its existing behaviour [35]. Modularisation allows the model to be extended and adapted to the evolving needs of the modeller and not to be short-lived. A module in Dymium can be simply viewed as a collection of related events. To facilitate and encourage development of new modules, Dymium provides helper functions that assists in module creation and impose a standard structure to make sure that it can be easily understood and reused by others. The list of modules that are being developed by us are listed in Table 1.

2.2.3. *Pipelining and scheduling of simulation events*

Events may occur at different time resolutions and a group of events may have a sequence in which they must be executed. This is what we call "pipelining and scheduling". It should be noted that Dymium operates in discrete time, as most urban models do.

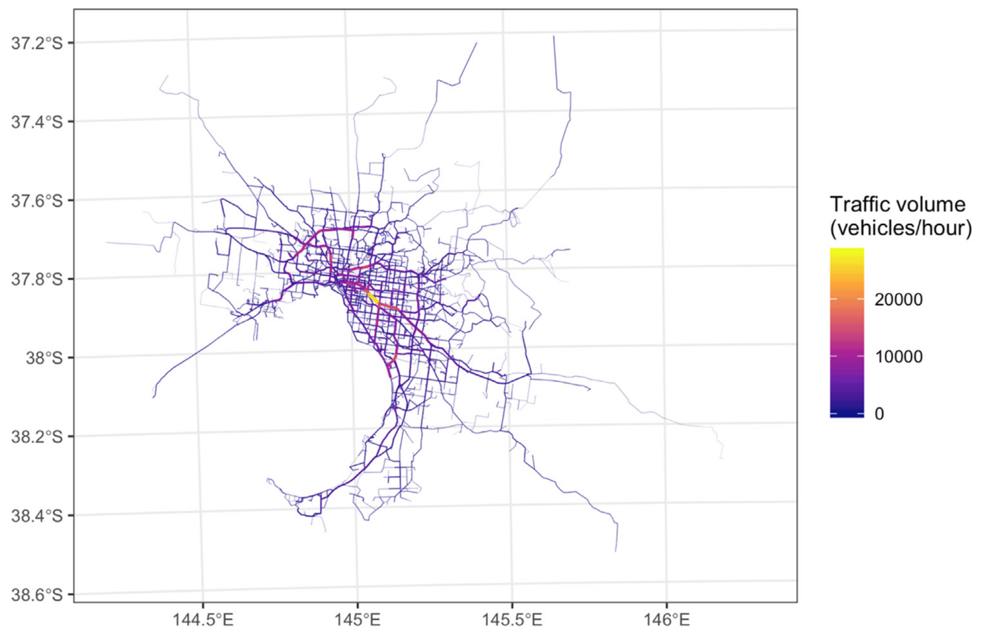

Fig. 6. Year 2030's forecasted average traffic volume (vehicles per hour) between 7 am to 8 am on an average day of the study area of the illustrative example (Melbourne, Australia). The traffic volume is scaled up 100 times from the 1% simulated travel demand.

It is possible to explicitly state the time steps in which an event should be triggered. The *Pipeline* class can be used to manipulate the order of events or trigger a set of event functions in a smaller time resolution, than the model's cycle length. For example, if the model's cycle length is one year, but some events normally occur every month, such as relocations of renters, those events can be configured to occur 12 times in one model cycle. You can think of this as a nested for-loop, where the most outer loop runs on a calendar year basis and the inner for loop goes by month.

2.3. A hello world example

Listing 1 shows an implementation of a simple population dynamic microsimulation model using Dymium. This model contains ageing, birth, and death events. The first part of the code is for data preparation and initialisation. While the second part shows a microsimulation pipeline encapsulated in a for loop set to repeat for 10 iterations. This subsection is meant to show some basic functions of Dymium. A more advanced use of the package is presented in the following section where we illustrate an example use case that utilises available Dymium modules.

3. Illustrative example

In this example, a hypothetical study of Melbourne, Australia, to forecast the evolution of population and urban traffic was set up. The demography module and the MATSim module from the dymiumModules repository were used. All demographic events, as shown in Table 2, only occur once every model cycle. The model cycle length is one year. The MATSim module provides an interface to MATSim [36] from R and also for transferring of the individual agents from Dymium to a MATSim's population file. In brief, MATSim is a state-of-the-art agent-based traffic simulation framework which uses daily activity patterns of agents to produce a highly detailed traffic outcome, unlike traditional four-step transport models. The MATSim model was run every five iterations since the overall difference in the traffic conditions of two consecutive years are relatively small and its running cost is relatively high compared to the demographic events. Hence, we scaled down the population and the network capacity to just 1% and also removed many small streets. All the parameters of the sub models used here are supplied in the supplementary file.

The World object contains around 37,000 Individual and 14,000 Household agents which is an equivalence of 1:100 of Melbourne population in 2011. The events are organised into a pipeline and scheduled to their appropriate time resolutions, as shown in Table 2. The simulation ran for 20 years from the base year 2011 to 2030. Fig. 5 shows the co-evolution of an urban population and temporal traffic pattern across a span of 20 years. The link statistics from MATSim can be easily made into a spatial plot with R to help the user to visually, see Fig. 6, identify potential bottlenecks that should be addressed in future transport planning of the study are.

4. Impact

Dymium is intended to make implementations of microsimulation models more accessible, reproducible, and reusable by a broad range of users. Being written in the R programming language should make this microsimulation framework more appealing to analysts and beginners as it is well supported with a wide range of packages that can help to accelerate their microsimulation workflows. With an increasing number of software packages that are specialised in different aspects of urban modelling, those packages, even in different programming environments, can be integrated Dymium, as example by the MATSim module. So far, the package has been used to develop a demographic microsimulation model for projecting the future population and household size of Melbourne, Australia [37]. The impact of microsimulation has been increasingly prominent, largely within academic research. Hence, having a microsimulation framework that is accessible, open source, free to use, and can be easily installed can relief users from technical difficulties. Enabling researchers to pursue new research questions more effectively and practitioners to put their plans into experimentation. With the official release of this package, we expect to see more researchers show their interest in Dymium as their microsimulation modelling toolkit.

```r
library(dymiumCore); library(data.table)
# create models, all individuals have 5% of dying and fertile females have 5% of giving birth
birth_model <- death_model <- list(yes = 0.05, no = 0.95)
# prepare individual data
ind_data <- data.table::copy(toy_individuals) %>%
            .[, .give_birth := "no"] # add a dummy column to store birth decision
# create an Individual object
ind <- Individual$new(.data = ind_data, id_col = "pid")
# create a World object
world <- World$new()
# add the Individual object to ‘world’
world$add(x = ind)
# create a pre-processing function
filter_fertile <- function(.data) {
  .data[age %between% c(18, 50) & sex == "female"]
}

# run the microsimulation pipeline for 10 iterations
for (i in 1:10) {
    world$set_time(i) %>%
        # ageing
        mutate_entity(entity = "Individual",
                      age := age + 1L) %>%
        # simulate giving birth
        transition(entity = "Individual",
                   model = birth_model,
                   attr = ".give_birth",
                   preprocessing_fn = filter_fertile) %>%
        # add newborns, by cloning children with age 0
        add_entity(entity = "Individual",
                   newdata = ind_data[age == 0, ],
                   target = .$entities$Individual$get_data()[.give_birth == "yes", .N]) %>%
        # reset the birth decision variable
        mutate_entity(entity = "Individual",
                      .give_birth := "no") %>%
        # simulate dying
        transition(entity = "Individual",
                   model = death_model,
                   attr = "age",
                   values = c(yes = -1L)) %>%
        # remove dead individuals
        remove_entity(entity = "Individual",
                      subset = age == -1) %>%
        # log the total number of alive individuals at the end of the iteration
        add_log(desc = "count:Individual",
                value = .$entities$Individual$get_data()[, .N])
}
```

Listing 1 – An implementation of a population dynamic microsimulation model using Dymium.

5. Conclusions and future development

We presented Dymium, a modular microsimulation modelling framework for integrated urban modelling in R. The main components and functionalities were discussed, and an example use-case was illustrated. There are many exciting features that are being actively developed by the main developers that will help with model calibration, a user-friendly GUI for modifying model parameters and visualise the simulation results, and ready-to-use modules that capture different aspects of the urban system. As an open source project, we welcome feedback and contributions from other researcher to improve the software and to make sure that Dymium is accessible to as many users as possible.

CRediT authorship contribution statement

Amarin Siripanich: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft, Visualization. **Taha Hossein Rashidi:** Conceptualization, Methodology, Writing - review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors acknowledge financial support from the Australian Research Council (DE170101346).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2020.100555.

References

[1] O'Donoghue C. Introduction. In: Handbook of microsimulation modelling. 2014, p. 1–21.
[2] Andreassen L. Demographic forecasting with a dynamic stochastic microsimulation model. 1993, Central Bureau of Statistics, http://www.ssb.no/a/publikasjoner/pdf/DP/dp_085.pdf (Accessed 8 September 2017).
[3] Birkin M, Wu B, Rees P, Wu B, Rees P. Moses: Dynamic spatial microsimulation with demographic interactions. In: New frontiers in microsimulation modelling. 2017, https://doi.org/10.4324/9781315248066-2.
[4] Blundell R, Duncan A, J. McC RAE, Meghir C. The labour market impact of the working families' tax credit. Fis Stud 2000;21:75–104, https://doi.org/10.1111/j.1475-5890.2000.tb00581.x.
[5] SOA. Dynacan: a canadian microsimulation model for public pension analysis. 1998, https://www.soa.org/Files/Research/Projects/Chapter_6.pdf.
[6] Harding A. APPSIM: The Australian dynamic population and policy microsimulation model. 2007, p. 11.
[7] Ballas D, Clarke G. GIS and microsimulation for local labour market analysis. Comput Environ Urban Syst 2000;24:305–30, https://doi.org/10.1016/S0198-9715(99)00051-4.
[8] Keegan MK. A dynamic microsimulation approach to simulating the impact of the labour force on issues relating to the ageing population. 2009, (n.d.) 227.
[9] Lymer S, Brown L. Developing a Dynamic Microsimulation Model of the Australian Health System: A Means To Explore Impacts of Obesity over the Next 50 Years. Epidemiology Research International; 2012, https://doi.org/10.1155/2012/132392.
[10] Rutter CM, Zaslavsky AM, Feuer EJ. Dynamic microsimulation models for health outcomes: A review. Med Decis Making 2011;31:10–8. http://dx.doi.org/10.1177/0272989X10369005.
[11] Shackleton N, Chang K, Lay-yee R, D'Souza S, Davis P, Milne B. Microsimulation model of child and adolescent overweight: making use of what we already know. Int J Obes 2019. http://dx.doi.org/10.1038/s41366-019-0426-9.
[12] Morris M, Kretzschmar M. A microsimulation study of the effect of concurrent partnerships on the spread of HIV in Uganda. Math Popul Stud 2000;8:109–33. http://dx.doi.org/10.1080/08898480009525478.
[13] Leclerc PM, Matthews AP, Garenne ML. Fitting the HIV epidemic in zambia: A two-sex micro-simulation model. PLOS ONE 2009;4:e5439. http://dx.doi.org/10.1371/journal.pone.0005439.
[14] Azevedo CL, Marczuk K, Raveau S, Soh H, Adnan M, Basak K, Loganathan H, Deshmunkh N, Lee D-H, Frazzoli E, Ben-Akiva M. Microsimulation of demand and supply of autonomous mobility on demand. Transp Res Rec 2016;2564:21–30. http://dx.doi.org/10.3141/2564-03.
[15] Goulias K, Kitamura R. Travel demand forecasting with dynamic microsimulation. Transp Res Rec 1992;1357:8–17.
[16] Waddell P. Urbansim: Modeling urban development for land use, transportation, and environmental planning. J Am Plann Assoc 2002;68:297–314. http://dx.doi.org/10.1080/01944360208976274.
[17] Fatmi MR, Habib MA. Microsimulation of life-stage transitions and residential location transitions within a life-oriented integrated urban modeling system. Comput Environ Urban Syst 2018;69:87–103. http://dx.doi.org/10.1016/j.compenvurbsys.2018.01.003.
[18] Miller EJ, Haroun PDA. A microsimulation model of residential housing markets, presented at. In: The 9th International association for travel behaviour research conference, GoldCoast, 2000.
[19] Kumar S, Kockelman K. Tracking size, location, and interactions of businesses: microsimulation of firm behavior in Austin, Texas. Transp Res Rec 2008;113–21.
[20] Maoh H, Kanaroglou P. Modelling firm failure: Towards building a firmographic microsimulation model. In: Employment Location in Cities and Regions. Berlin, Heidelberg: Springer; 2013, p. 243–61. http://dx.doi.org/10.1007/978-3-642-31779-8_12.
[21] Chingcuanco F, Miller EJ. A microsimulation model of urban energy use: Modelling residential space heating demand in ILUTE. Comput Environ Urban Syst 2012;36:186–94. http://dx.doi.org/10.1016/j.compenvurbsys.2011.11.005.
[22] Moeckel R. Constraints in household relocation: Modeling land-use/transport interactions that respect time and monetary budgets. J Transp Land Use 2016;10. http://dx.doi.org/10.5198/jtlu.2015.810.
[23] Marshall D. CAPITA-B: A behavioural microsimulation model, canberra. 2016, https://www.pc.gov.au/research/supporting/capita-b/capita-b.pdf (Accessed 21 February 2019).
[24] van Sonsbeek J-M, Alblas R. Disability benefit microsimulation models in the Netherlands. Econ Modell 2012;29:700–15. http://dx.doi.org/10.1016/j.econmod.2012.01.004.
[25] Waddell P. Towards a behavioral integration of land use and transportation modelling. In: International conference on travel behaviour research, 9th, 2000, Gold Coast, Queensland, Australia, Gold Coast, Queensland, Australia, 2000.
[26] Moeckel R, Garcia CL, Chou ATM, Okrah MB. Trends in integrated land use/transport modeling: An evaluation of the state of the art. J Transp Land Use 2018;11. http://dx.doi.org/10.5198/jtlu.2018.1205.
[27] Miller EJ. Agent-based activity/travel microsimulation: what's next? In: The Practice of Spatial Analysis. Springer; 2019, p. 119–50.
[28] Miller EJ. Viewpoint: Integrated urban modeling: Past, present, and future. J Transp Land Use 2018;11. http://dx.doi.org/10.5198/jtlu.2018.1273.
[29] Lee DB. Requiem for large-scale models. J Am Instit Plann 1973;39:163–78. http://dx.doi.org/10.1080/01944367308977851.
[30] O'Donoghue C, Dekkers G, et al. Increasing the impact of dynamic microsimulation modelling. Int J Microsimul 2018;11:61–96.
[31] Chang W. R6: encapsulated classes with reference semantics. 2019, https://CRAN.R-project.org/package=R6.
[32] Dowle M, Srinivasan A. Data.table: Extension of 'data.frame'. 2019, https://CRAN.R-project.org/package=data.table.
[33] Bengtsson H. Future: Unified parallel and distributed processing in r for everyone. 2020, https://CRAN.R-project.org/package=future.
[34] Li J, O'Donoghue C. Evaluating binary alignment methods in microsimulation models. J Artif Soc Soc Simul 2014;17. http://dx.doi.org/10.18564/jasss.2334.
[35] Cassells R, Harding A, Kelly S. University of Canberra National Centre for Social and Economic Modelling, Problems and Prospects for Dynamic Microsimulation: A Review and Lessons for APPSIM, NATSEM. Bruce, A.C.T.: University of Canberra; 2006.
[36] Horni A, Nagel K, Axhausen KW, eds. The Multi-Agent Transport Simulation MATSim. Ubiquity Press; 2016, http://dx.doi.org/10.5334/baw.
[37] Siripanich A, Rashidi T. A demographic microsimulation model with an integrated household alignment method. 2020, arXiv:2006.09474 (Accessed 18 June 2020).