R and Python Annotation Bindings for OMS

Francesco Serafin

James D. Westervelt

Charles R. Ehlschlaeger

Olaf David

Liqun Lu

See next page for additional authors

Presenter/Author Information
Francesco Serafin, James D. Westervelt, Charles R. Ehlschlaeger, Olaf David, Liqun Lu, Antoine Petit, Zhoutong Jiang, and Yanfeng Ouyang

iEMSs International Congress on Environmental Modelling and Software

9th International Congress on Environmental Modelling and Software
Fort Collins, Colorado, USA, Mazdak Arabi, Olaf David, Jack Carlson, Daniel P. Ames (Eds.)
https://scholarsarchive.byu.edu/iemssconference/2018/

R and Python Annotation Bindings for OMS

Serafin F.[1A], Westervelt J.D.[2B], Ehlschlaeger C.R.[3B], David O.[4C], Liqun L.[5D], Petit A.M.A.[6D], Zhoutong J.[7D], Yanfeng O.[8D]
[1]francesco.serafin@unitn.it, [2]westerve@comcast.net, [3]Charles.R.Ehlschlaeger@usace.army.mil, [4]odavid@colostate.edu, [5]liqunlu2@illinois.edu, [6]apetit@illinois.edu, [7]zjiang30@illinois.edu, [8]yfouyang@illinois.edu
[A]University of Trento, [B]CERL US Army Engineer Research and Development Center, [C]Colorado State University, [D]University of Illinois

Abstract: OMS3 is an environmental modeling framework designed to support and simplify the development of scientific environmental models. It is implemented in Java, a programming language that allows the framework to be flexible and non-invasive. Consequently, Java is the native language for developing OMS-compliant components. However, OMS3 aims to ensure the longevity of old model implementations by providing C/C++ and Fortran bindings that allow for connecting slightly modified legacy environmental software to newly developed Java components. In the recent years, three scientific programming languages drew the modeling community's attention: R, Python, and NetLogo. They have a flat learning curve, numerous scientific libraries, and duck typing makes them an attractive solution for fast scripting. Furthermore, they have an active developer community that keep releasing and improving open source scientific packages. This is a relevant aspect when it comes to facilitating and speeding up the implementation of scientific algorithms. Therefore, OMS3 integration capabilities have recently been enhanced to provide R, Python, and NetLogo bindings. As a result, multi-language modeling solutions can be tailored to meet the scientific community's needs. Thanks to the framework's non-invasiveness, R, Python and NetLogo scripts must only be slightly modified with source code annotations to become OMS-compliant components. The resulting components are nevertheless still executable from within the original environments. This contribution shows two actual applications of the implemented R and Python bindings, the NetLogo implementation is not addressed in this paper. The Regional Urban Growth (RUG) is implemented in R and the TRansportation ANalysis SIMulation System (TRANSIMS) models require the Python Run Time Environment (RTE) module to run. The RUG model is a landscape model capable of evaluating impacts of new regional urban development on surrounding environment and projecting long-term growth-management plans. TRANSIMS is a software suite based on a cellular automata microsimulator which performs regional transportation system analyses. Both model suites are among OMS enabled models for the FICUS project, the "Framework for Integrating the Complexity of Uncertain Systems". Furthermore, the model application flexibility was enhanced by introducing Docker containers in the workflow to alleviate the burden of complex software management and setup.

Keywords: *OMS3; R; Python; RUG; TRANSIMS, FICUS*

1 INTRODUCTION

OMS3 is a flexible and non-invasive environmental modeling framework (David et al. 2013, Lloyd 2011). Its main objective is to simplify environmental model development by streamlining the translation of physical processes into programming algorithms. It allows for encapsulating each algorithm into a standalone component ensuring the "single responsibility" principle. It lowers the development effort related to data reading and writing, data analysis and visualization, component interaction, temporal-spatial stepping and multi-threading/multi-processor computations. As a result scientists can focus on scientific understanding of environmental phenomena rather than software development.

OMS3 is Java-based, and therefore Java components are natively supported. In order to maintain compatibility with legacy Fortran and C/C++ software, OMS3 leverages the use of native shared libraries and provides Fortran and C/C++ bindings. However, the modeling community's use of scripting/programming languages like R, Python and NetLogo is rapidly taking off. These languages are easy to learn and use because of their friendly syntax and semantics. They rely on user and developer communities, which share on-line implementation and problems solutions, generic information and most importantly well designed scientific packages. Some notable Python examples are NumPy (Oliphant 2006) and SciPy (Jones et al. 2014). Some notable R examples are gstat (Pebesma et al. 1998), raster (Hijmans et al. 2017) and randomForest (Liaw et al. 2002).

Scientists and engineers solely want to focus on solving their research questions and problems. These scripting languages are consequently very attractive and proper OMS3 bindings have become necessary. The main concern while developing OMS bindings was to keep the user experience in setting up Python and R OMS-compliant components as close as possible to OMS Java component development. Section 2 is focused on describing the user approach in modifying Python and R scripts into OMS-compliant components. Section 3 describes actual framework side implementation of both bindings while section 4 introduces the process of bundling OMS3 into a Docker image. Section 5 shows two actual applications: the R-based Regional Urban Growth (RUG) model (Westervelt et al. 2011) and the Python wrapped TRansportation ANalysis SIMulation System (TRANSIMS) model (Smith et al. 1995). Section 6 provides concluding remarks and identifies current constraints and needs for future development.

Moreover, OMS3 was recently bundled into a Docker (Merkel 2014) image to further simplify user experience: once Docker is installed on the machine, no further software installation and library linking are required to run OMS3. A user needs to provide only a properly set up OMS3 project. The Docker container then takes care of building the project and running the modeling solution, automatically connecting every type of component.

2 USER EXPERIENCE

An OMS component is basically a plain Java class with framework metadata annotations. Input/output variables are listed as fields and annotated with *@In* and *@Out* OMS annotations. The one mandatory method with an *@Execute* annotation encapsulates the main algorithm and calls related methods or objects. Two more methods can be annotated with *@Initialize* and *@Finalize* and are respectively executed before and after the entire simulation. They are optional methods, though. A user may also add further optional annotations to capture comments and component design ideas into metadata for generating documentation later or perform tests.

These basic concepts were used in the design of both Python and R bindings. Accordingly, two main development steps were identified to seamlessly adapt Python or R scripts into OMS-compliant components:

1. Determine the function encapsulating the main algorithm if the script is already split into functions, otherwise wrap the entire script into one main function;
2. Identify input and output variables and list them at the very beginning of the script.

Then, suitable annotations have to be accomodated. Figure 1 and Figure 2 ease the understanding of this simple but crucial step: Figure 1 shows the annotated code snippet of the R component *AttractorAnalysis.R*, which is part of the RUG model; Figure 2 illustrates the annotated code snippet of the Python component *TransimsObj.py*, which is the Python wrapper for executing and connecting TRANSIMS executables.
A couple of similarities can be underlined in Figure 1 and Figure 2: annotations are hidden in comments; Java data types are explicitly specified right after *@In* and *@Out* annotations.

The first aspect allows for maintaining compatibility of scripts with their original interpreters. To execute the OMS-compliant scripts from within their original environments, user is asked to: 1) assign input values to each input variable and null values to each output variable (or just comment them to avoid parsing errors); 2) call the main function to execute the script.

The second aspect takes into account the absence of declared data types in both Python and R. Thus, Java equivalent types must be defined between parentheses right after the annotation to allow for proper conversions when R or Python components are connected to Java or Fortran or C/C++ components.
Figure 1 shows how a stack of raster maps (*masterRaster*, line 5), a list of raster maps (*interconnectMaps*, line 8) and a list of strings (*instructions*, line 11) are fed to the *AttractorAnalysis.R* component. After the proper computation, the raster map describing the attractiveness of strategic locations in the study area (*attractorMap*, line 14) is returned. Figure 2 shows how the path to the directory gathering TRANSIMS modules (*BINDIR*, line 6), the path to the working directory (*PROJECT*, line 8), the name of the TRANSIMS module (*executable*, line 10) and name of the related file of input data and parameters (*controlFile*, line 12) are inputs to the *TransimsObj.py* component. When the run is over, the component returns the proper message (*simDone*, line 14).

```
1   library(raster)
2   library(doParallel)
3
4   # @In("CoverageStack")
5   masterRaster
6
7   # @In("List<GridCoverage2D>")
8   interconnectMaps
9
10  # @In("List<String>")
11  instructions
12
13  # @Out("GridCoverage2D")
14  attractorMap
15
16  # @Execute
17  main <- function() {
18      # RUG attractor analysis
19      # ...
20      attractorMap <<- calcAttractorMap()
21  }
```

Figure 1. R OMS-compliant version of the AttractorAnalysis.R

```
1   import os
2   import sys
3   from TransimsRTE import *
4
5   # @In("String")
6   BINDIR
7   # @In("String")
8   PROJECT
9   # @In("String")
10  executable
11  # @In("String")
12  controlFile
13  # @Out("String")
14  simDone
15
16  # @Execute
17  def execute():
18      # Transims OMS object
19      # ...
20      global simDone
21      simDone = "Transims obj processed"
```

Figure 2. Python OMS-compliant version of the TransimsObj.py

Currently, only standard data type matching is available. The R binding temporarily provides an inner matching of “Raster”, “List of Raster” and “CoverageStack” between the raster R package and the Geotools Java library. However, a plug-in system of data type conversions is under development. The purpose is to allow each user to implement the proper conversion between data types, and sharing it with the entire community. Nevertheless, two connected R or Python components can share generic *Object* data type which does not require any matching (see Table 1 and Table 2 for available data types conversions).

Java data type	R data type	Python data type
int	int	int
double	double	double

String	String	String
int[]	vector of int	*jarray(...,JINT_ID,...), from jep import jarray, JINT_ID
double[]	vector of double	*jarray(...,JDOUBLE_ID,...), from jep import jarray, JDOUBLE_ID
String[]	vector of String	*jarray(...,JSTRING_ID,...), from jep import jarray, JSTRING_ID
GridCoverage2D	raster	
CoverageStack	RasterStack	
List<GridCoverage2D>	list() of Raster	
Object	Object	Object
List<Integer>		[]
List<Double>		[]
List<String>		[]
List<Object>		[]
Map<Object, Object>		dictionary

Table 1. R and Python available data types.
***Python binding makes use of **jarray** instead of **Numpy** data structures because jarray makes data transfer faster and more efficient for the back-end Jep.*

One design aspect relates to both bindings: in order to actually fill output variables and avoid declaring local function variables, a user must make use of specific operators. In R scripts, output variables must be assigned using the *double arrow assignment operator* <<- which allows for modifying variables in a parent level (e.g. *attractorMap* in line 20, Figure 1). In Python scripts, output variables must be declared *global* at the very beginning of the main function to allow for modifying variables at parent level (e.g. *simDone* at line 20, Figure 2). Output variables have to be declared outside the main function as well (e.g. line 14, Figure 1 and Figure 2). In this way, OMS3 can access their content and perform proper connections with other components.
With respect to framework invasiveness, no specific OMS3 or other APIs have to be imported or extended.

3 TECHNICAL APPROACH AND IMPLEMENTATION

To provide for a smooth user experience the actual implementation burden is moved into the framework. Python and R are both cross-platform, interpreted, high-level scripting and programming languages. Thus, they both require interpreters to parse and execute a script. Simple access through shared libraries like Fortran or C/C++ through JNI (Gordon 1998) does not work. Consequently, OMS3 needs to directly intercommunicate to R and Python interpreters.

The common approach implies the generation of a Java OMS component aiming to wrap a single R or Python script while building the OMS3 project. Eventually OMS3 calls only Java classes. When it is time to run the Java wrapper, this starts a connection to R or Python environment, sends the script to get parsed by the proper interpreter, sends input data and retrieves output information. It provides also for properly converting input/output standard data types or data structures between languages. Obviously, R and Python environments have to be already installed on the machine and correctly linked to OMS3.

3.1 R back-end: Rserve

Rserve is a TCP/IP server developed by Urbanek S. (2003) to leverage R functionalities from within different programming languages. It was developed following three important design principles: separation of the R system from the application, flexibility for leveraging most R facilities and speed to have a performant client-server communication. However, the most interesting feature is the management of multiple clients simultaneously. Rserve creates a different data space and working directory for each new connection. Because each R OMS-compliant component opens a new independent connection to the R environment, multiple R OMS-compliant components can be executed in parallel without interference. This allows for leveraging OMS3 implicit multithreading computation.

Rserve requires installation of an R interpreter and the Rserve package on a local computer to properly work with OMS3.

3.2 Python back-end: Jep

Jep is an open source Python package (https://github.com/ninia/jep) that leverages both JNI and CPython API to run a Python interpreter from within the Java Virtual Machine (JVM). Its main feature is that of creating a different sandboxed sub-interpreter for each new Jep instance. In this way concurrent sub-interpreters don't share imported modules or global variables, thus avoiding conflicts.

To properly exercise Jep from within OMS3, the Jep package has to be installed in addition to the proper Python interpreter. This is not trivial on Windows OS which requires an additional installation of a dedicated build tool. Furthermore, Jep shared libraries have to be accurately linked to the correct environmental variable (e.g. LD_LIBRARY_PATH) to be accessible by the Java process. Switching between Python2 and Python3 might be confusing and error prone as well.

4 DOCKER IMAGE BUNDLE

As explained in sections 3.1 and 3.2, Rserve and Jep require installation of a proper R or Python environment and accurate linking of involved libraries, Jep especially. But this means that a user is expected to take care of software installation and required libraries, which are both OS specific and require some OS proficiencies. This is diametral to the OMS3 principle of simplifying user experience by separating responsibilities between users and software developers. To overcome this constraint a recently released technology has been leveraged and OMS3 has been bundled into a Docker image.

Docker is a software system that packages a software application and its dependencies into an image. It then runs that image as a virtual container on top of a host OS. It is similar to a virtual machine (VM) since it isolates the running process of bundled applications from interfering with running processes of the host OS. However, container virtualization is more lightweight than a Hypervisor based VM. It virtualizes at operating-system-level without the needs of a hypervisor, which is an additional software on top of the host OS to create, run and manage virtual machines(Merkel 2014). Docker images are platform independent. Consequently, the same Docker containers run on every OS once Docker is properly installed.

A Docker image results from a build process that starts off from a Dockerfile. The latter contains instructions required to install and setup applications along with dependencies. It also contains instructions for proper library linking and environment variables set up. The latter don't interfere with environment variable of the host operating system because Docker isolates the bundled application from the hosting OS. To correctly exercise an OMS modeling solution the OMS project is mounted into the running Docker container. The OMS3 image is made available at https://hub.docker.com/r/omslab/oms/ and Dockerfiles are made available at https://github.com/sidereus3/oms-docker.

Both, R and Python rely on hundreds of packages which cannot be included into a Docker image for the sake of size limits and the impossibility of continuous updates when new packages are released. To overcome this constraint two slightly different approaches have been implemented.

4.1 OMS R packages management

User scripts normally import standard and locally installed R packages through the *library()* command. The Docker image manages linking of bundled R environment to the additional *Rlibs/build/* folder.

The OMS Docker image provides a feature that allows for automatically downloading and building R packages required by R scripts in the OMS project. This is a one-time process which is enabled during OMS project build. When specific R packages are required, the user is asked to create a *Rlibs* folder inside the main OMS project. The user has to provide a file named *package.txt* with a list of names of required packages, located in *Rlibs*. During the building step, the Docker container looks for *Rlibs* folder. If it exists and contains the file *package.txt*, the container reads all the listed packages and builds the dependency tree. Then it starts downloading source code of each package into *Rlibs/source/* creating a local R package repository. As a final step, the Docker image goes through the repository, and builds and installs each package into *Rlibs/build/*.

Because Docker is platform independent the OMS project can be zipped and moved to a different machine. If the version of the Docker image does not change, the transferred OMS projects can be directly executed.

4.2 OMS Python packages management

The OMS Docker image does not currently provide any tool for automatically downloading required python packages and related dependencies. However, if the user provides Python packages within the folder *Pylibs/* in the main project directory, the OMS Docker image automatically makes new modules and packages available for standard import.

5 APPLICATIONS

The development of both R and Python binding has been continuously tested with two actual models in order to gain experience and drive the development direction from the very beginning. The R binding was tested using the RUG model, while the Python binding was tested with the TRANSIMS model.

5.1 RUG model

The Regional Urban Growth (RUG) model evaluates the attractiveness of a specific location with respect to urban growth. It is a raster based model: input data is a landscape raster map which allows for estimating development attraction on each location depending on proximity to development attractors (roads, highways, etc.) (Westervelt et al. 2011). The RUG model was a stand-alone, well implemented R software that leverages availability of R packages like raster (Hijmans et al. 2017), doParallel (Calaway et al. 2015), randomForest (Liaw et al. 2002) and gdistance (van Etten 2017). To make this model OMS-compliant, it was split into three different components: Travel Time Analysis, Development Analysis and Attractor Analysis. This partitioning made possible to identify functions containing the main algorithms and input/output data.

The RUG model performs a raster based analysis, and thus two Java components for raster reading and writing were implemented leveraging Geotools APIs. Proper mappings for *Raster*, *List of Rasters* and *CoverageStack* data structures between Java and R (and vice versa) were included in the R binding. The final modeling solution is illustrated in Figure 3.

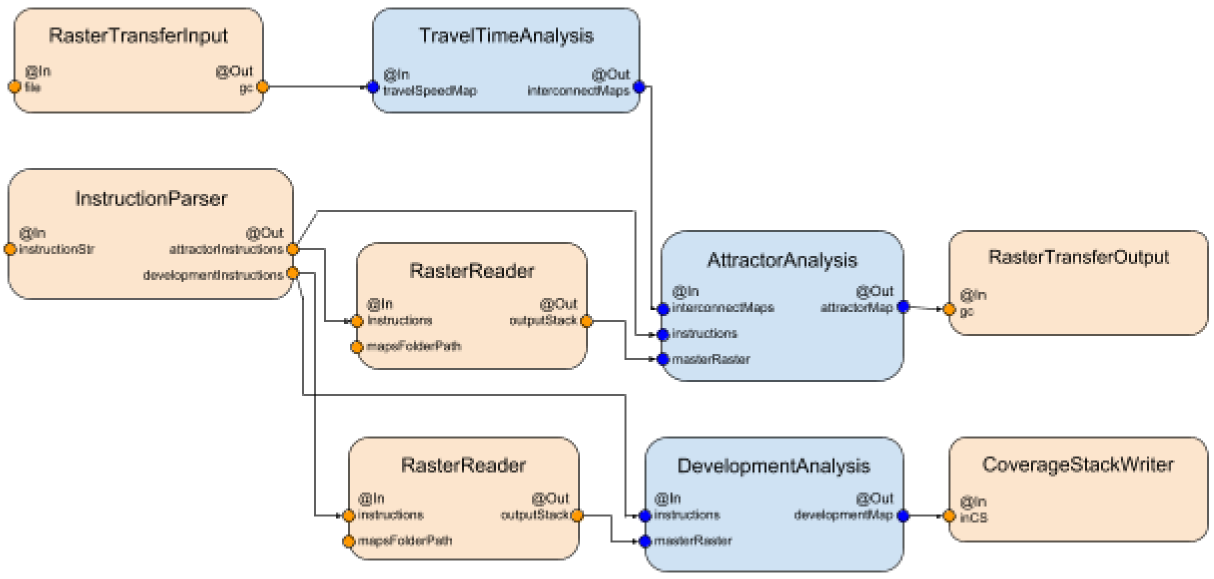

Figure 3. RUG modeling solution: Java components in light orange, R OMS-compliant components in light blue.

5.2 TRANSIMS model

The TRansportation ANalysis and SIMulation System (TRANSIMS) model evaluates integrated regional transportation systems. Regional population of individual travelers and freight loads with travel activities and travel plans are core of modeling computation (Smith et al. 1995). TRANSIMS is more a set of tools than a homogeneous model. Each module is a stand-alone C++ program, which builds into a separate, statically linked executable.

A Python Module for encapsulating TRANSIMS executables has been recently released. TRANSIMS RTE (Run Time Environment) improves scripting flexibility providing for easy modeling solution design. It allows for setting up TRANSIMS keywords, e.g. *@NEW* and *@OLD*, and running a proper executable and related control file from within a Python script. Because a TRANSIMS modeling solution is a sequence of calls to different modules, a generic TRANSIMS-OMS component has been abstracted from a Python script. A simple Java class reads a csv file with a list of executable names and related control files and the feeds the TRANSIMS-OMS component while the list is empty. A sample modeling solution is shown in Figure 4.

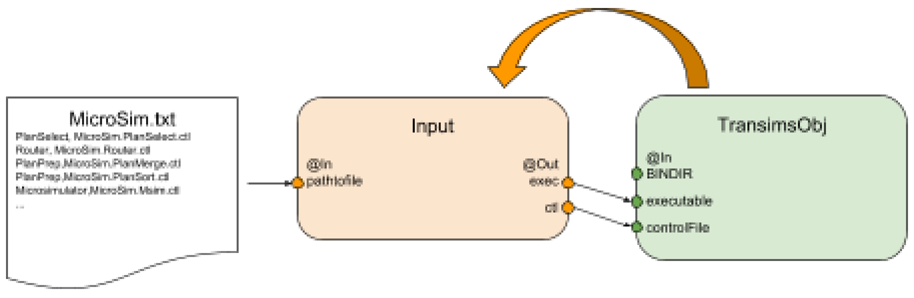
MicroSim.txt
PlanSelect, MicroSim.PlanSelect.ctl
Router, MicroSim.Router.ctl
PlanPrep,MicroSim.PlanMerge.ctl
PlanPrep,MicroSim.PlanSort.ctl
Microsimulator,MicroSim.Msim.ctl
...
Input
@In
pathtofile
@Out
exec
ctl
TransimsObj
@In
BINDIR
executable
controlFile

Figure 4. TRANSIMS sample modeling solution: Java component in light orange, Python OMS-compliant component in light green.

6 CONCLUSIONS

This paper shows how two of the most notable and widely used programming languages in the scientific community have been integrated into OMS3. It can be concluded that the process of

Python/R scripts adaptation into OMS-compliant components is straightforward and doesn't require user specific proficiency in understanding mixed language programming. This opens a future perspective for easily creating multi-language modeling solutions, that leverages already available scientific packages and avoid code duplication.

Thank to the innovative technology of Docker containers, a user does not experience the burden of connecting OMS3 with Python and R interpreters. An automated process for R package retrieval and building is provided in the Docker image. The two presented applications demonstrate the applicability and relevance. The implementation aims for design consistency with existing annotation based representation of components.

However, some limitations still exist and will be addressed in future developments: a fully flexible mapping of R/Python into Java data structures is not yet available; automated process for Python packages retrieval is not provided; and only the latest version of a deployed R package is retrieved, user cannot automatically download a specific package version.

REFERENCES

Calaway, R., Weston, S., Tenenbaum, D. and Analytics, R., 2015. doParallel: Foreach parallel adaptor for the 'parallel' package. *R package version*, *1*(10).

David, O., Ascough II, J.C., Lloyd, W., Green, T.R., Rojas, K.W., Leavesley, G.H. and Ahuja, L.R., 2013. A software engineering perspective on environmental modeling framework design: The Object Modeling System. *Environmental Modelling & Software*, *39*, pp.201-213.

Gordon, R., 1998. *Essential JNI: Java Native Interface*. Prentice-Hall, Inc..

Hijmans, R.J., van Etten, J., Cheng, J., Mattiuzzi, M., Sumner, M., Greenberg, J.A., Lamigueiro, O.P., Bevan, A., Racine, E.B., Shortridge, A. and Ghosh, A., 2017. Package 'raster'.

Jones, E., Oliphant, T. and Peterson, P., 2014. SciPy: open source scientific tools for Python.

Liaw, A. and Wiener, M., 2002. Classification and regression by randomForest. *R news*, *2*(3), pp.18-22.

Lloyd, W., David, O., Ascough II, J.C., Rojas, K.W., Carlson, J.R., Leavesley, G.H., Krause, P., Green, T.R. and Ahuja, L.R., 2011. Environmental modeling framework invasiveness: Analysis and implications. *Environmental modelling & software*, *26*(10), pp.1240-1250.

Merkel, D., 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), p.2.

Oliphant, T.E., 2006. *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.

Pebesma, E.J. and Wesseling, C.G., 1998. Gstat: a program for geostatistical modelling, prediction and simulation. *Computers & Geosciences*, *24*(1), pp.17-31.

Smith, L., Beckman, R., Anson, D., Nagel, K. and Williams, M., 1995. *TRANSIMS: Transportation analysis and simulation system* (No. LA-UR-95-1664; CONF-9504197-1). Los Alamos National Lab., NM (United States).

Urbanek, S., 2003. Rserve--a fast way to provide R functionality to applications. In *PROC. OF THE 3RD INTERNATIONAL WORKSHOP ON DISTRIBUTED STATISTICAL COMPUTING (DSC 2003), ISSN 1609-395X, EDS.: KURT HORNIK, FRIEDRICH LEISCH & ACHIM ZEILEIS, 2003 (HTTP://ROSUDA. ORG/RSERVE*.

van Etten, J., 2017. R package gdistance: distances and routes on geographical grids.

Westervelt, J., BenDor, T. and Sexton, J., 2011. A technique for rapidly forecasting regional urban growth. *Environment and Planning B: Planning and Design*, *38*(1), pp.61-81.