Article

B Planning and Design

Environment and Planning B:
Planning and Design
0(0) 1–22
© The Author(s) 2016
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0265813516647062
epb.sagepub.com
SAGE

Integration of land use, land cover, transportation, and environmental impact models: Expanding scenario analysis with multiple modules

Harutyun Shahumyan
University of Maryland, USA

Rolf Moeckel
Technical University Munich, Germany

Abstract

It is an expensive and time-consuming task to develop a new model. Furthermore, a single model often cannot provide answers required for complex decision making based on multiple criteria. Coupling models are often applied to make use of existing models and analyze complex policy questions. This paper provides an overview of possible model integration approaches, briefly explains the modules that were integrated in a particular application, and focuses on the integration methods applied in this research. While the initial attempt was to integrate all models as tightly as possible, the authors developed a much more agile integration approach that allows adding and replacing individual modules easily. Python wrappers were developed to loosely couple land use, land cover, transportation, and emission models developed in different environments. ArcGIS Model Builder was used to provide a graphical user interface and to present the models' workflow. The suggested approach is especially efficient when the models are developed in different programming languages, their source codes are not available, or the licensing restrictions make other coupling approaches impractical.

Keywords

Model integration, coupling, land use, transportation, emission, Geographic Information System (GIS)

Introduction

Policy makers are facing challenges of managing multifaceted urban and environmental systems influenced by global factors—such as population growth, migration, recession,

Corresponding author:
Harutyun Shahumyan, University of Maryland, 1112J Preinkert Field House, College Park, MD 20742, USA.
Email: shahumyan@hotmail.com

climate change—as well as by local actors, such as parties or companies who direct the development according to their own interests (Geertman and Stillwell, 2009). Confronted with such complexity, decision makers need adequate tools to better understand and evaluate the effects of policy interventions in urban regions. Such pressure already led to the development of numerous models covering different discipline-specific areas. However, the interconnected character of human and natural systems, such as demographics, transportation, infrastructure, economics, agriculture, land cover, climate, air, and water, requires an integrated approach in decision making and policy-impact assessment, and consequently in modeling (Argent, 2004; Laniak et al., 2013; Schmitz et al., 2009; Van Ittersum et al., 2008). Furthermore, integration may have a horizontal and vertical dimension (Moeckel et al., 2015). The horizontal dimension commonly refers to the integration of various domains (such as environment, land use, transportation, etc.), while the vertical integration refers to various geographic layers that need to be integrated (such as urban, regional, and national scales).

Nevertheless, model integration faces some scientific as well as technical challenges (Argent, 2004; Belete et al., 2014; Van Delden et al., 2011). The main scientific challenges relate to dealing with different domains, paradigms, assumptions, scales, spatial, and temporal resolutions used by different models. Technical challenges include implementing the software integration of the models, providing dynamic feedback loops, managing data exchange and storage, and developing relevant user interfaces (Lam et al., 2004). Moreover, as Voinov and Shugart (2013) emphasize, treating models only as software in solving the integration challenge may result in perfectly valid software products, which however, are diverging from the theories behind and are practically useless as models.

The integration is especially difficult if the models are developed independently without any built-in method for linking to other models. Moreover, they often are developed in different programming languages and software environments and may have various licensing restrictions. The task becomes even more complicated with proprietary models or when the access to a model's source code is limited. The authors explored diverse coupling approaches and evaluated their applicability for the specific modeling requirements. However, the existing couplers have been developed with different objectives and constraints in mind. None of the couplers identified by the authors were suitable for the defined integration task. Therefore, an alternative approach of loose model coupling was applied. The suggested approach is especially efficient when the models are developed in different programming languages, their source codes are not available, or if the licensing restrictions or limited resources make other coupling approaches infeasible. The presented approach offers a viable solution to couple such models without the need to change or even access to the sources codes. The approach was applied in a case study to couple transportation, land use, land cover, and environmental impact models, which were developed and are continuing to develop independently in different organizations. Therefore, keeping them separate and not changing the source codes were an important aspect for keeping the integrated suite compatible with the continuously updated versions of the selected models.

Methodology

This section identifies requirements for model integration and reviews coupling approaches described in the literature. Being particularly relevant for this research, model integration options with GIS and Python are reviewed in more detail.

Key requirements

Taking into account the overall goals of this research, the reviewed literature and the type of models considered, the following key requirements have been identified for the model integration approach:

- Ability to develop models independently and easily connect them with other models;
- A modular approach supporting reusability of components and adding new components;
- Minimal or no change in source codes of the models;
- Capacity to link models developed in different programming languages;
- Ability to deal with different licensing requirements;
- Avoid all manual data transfer;
- User-friendly graphical interface;
- Compatibility with GIS for easy data visualization and spatial analysis;
- Adequate run time;
- Minimal costs and efficient timing for implementation.

The following review of approaches described in the literature is analyzed in particular with regard to these requirements.

Model coupling approaches

The general definition of model coupling implies that originally independent model processes interact. The implementation of such interaction, however, can vary depending on the project goals, available resources, the models' specifics, requirements, and limitations. For the coupling of environmental models, Brandmeyer and Karimi (2000) developed a five-level coupling hierarchy, which includes manual data transfer, loose coupling, shared coupling, joined coupling, and tool coupling.

The *manual data transfer* method is the most basic level of model coupling, which includes manual extraction, transfer, and conversion of output produced by one model to be used as an input by other models. Though this approach requires minimal initial cost and time to apply, it is not convenient when multiple runs and frequent data exchange are required (Brandmeyer and Karimi, 2000). Manual data transfer is also error prone, often making it impossible to trace back at a later point if data were transferred correctly.

The data exchange between models is automated in *loose coupling*. Models, though, still work independently and the user interacts with each model separately (Wong et al., 2009). Loose coupling also has low initial cost, requires minimal changes to existing codes, and the models still can be developed independently. However, if data structures change in any of the linked models, data conversion mechanisms between the effected models require particular attention.

In *shared coupling,* the models either share a user interface or the data storage. For the first approach, a single user-friendly interface hides the internal coupling method, making it less confusing (Berry et al., 1997). In data coupling, the models are kept separate but share the data storage (Van Walsum and Veldhuizen, 2011). Shared user-interface coupling supports proprietary models and reduces the time the user needs to interact with the models. However, a user interface update often is required when models are updated. Data coupling makes data maintenance simpler. However, the overall model interface and performance depend on the database management system used (Brandmeyer and Karimi, 2000).

Joined coupling employs both the common user interface and data storage and may use two structurally different approaches: embedded coupling, where one model contains another (Liu et al., 2014); and integrated coupling, where each model is a peer of every other model (Sudicky et al., 2003). Joined coupling reduces the development costs and promotes code reusability. But it requires access to the models' source codes and a single operating system (OS).

For *tool coupling*, the models are coupled using a modeling framework (Babendreier and Castleton, 2005; Moore and Tindall, 2005). This supports community model development and can be used with both legacy and new models. Though it has higher initial costs due to framework design and development, several such tools have been developed, such as the Open Modeling Interface (OpenMI) (Gregersen et al., 2007), the Model Coupling Toolkit (Warner et al., 2008), the Community Surface Dynamics Modeling System (CSDMS) (Overeem et al., 2013), the Earth System Modeling Framework (Hill et al., 2004), the Framework for Risk Analysis Multimedia Environmental Systems (FRAMES) (Shah et al., 2004; Whelan et al., 2014),PCRaster (Schmitz et al., 2009), Open Projet d'Assimilation par Logiciel Multimethodes (O-PALM) (Piacentini et al., 2011), OASIS (Valcke, 2013), and Integrated Component Modelling System (ICMS) (Rahman et al., 2004). However, each of these tools has their specific requirements regarding OS, programming languages, data format, access to the source code, model licenses, and so on. They often demand changes, adaptation, or rewriting the model source codes, which usually requires programming and data/language interoperability expertise.

The order of running the models and the data feedback frequencies also effect on model coupling choices. The "sequential" coupling scheme provides the weakest form of the integration, where the first model runs the required time step/period and provides the output to the second model, which only runs after getting the results of the first model (Van Walsum and Veldhuizen, 2011). This scheme is often used for manual data transfer or loose model coupling. A drawback of such an approach is that the stability between the two linked models is determined by the model that gets updated first, which can lead to inconsistencies for the second model. In contrast, the "fully coupled" scheme supports the full feedback between models within each time step. However, full coupling usually requires code modification to organize such feedback. Moreover, it may result in iterations within iterations and increase its computational load essentially, reducing the overall efficiency (Van Walsum and Veldhuizen, 2011).

Each of the described approaches has its advantages and disadvantages (Brandmeyer and Karimi, 2000; Droppo et al., 2010) and the selection of the method mainly depends on the model requirements, research goals, and available resources.

Model integration through GIS

Combining spatial analysis and mapping tools with a database management system makes GIS an effective platform for bringing modeling tools together with spatial and tabular data (Kittle et al., 2006). Considerable efforts have been made to use GIS as an integration tool for different environmental models including soil erosion (Brazier et al., 2005), land use (Clarke and Gaydos, 1998), hydrologic (Devantier and Feldman, 1993), water quality (León et al., 2002), pollution and watershed (Basnyat et al., 2000; Kittle et al., 2006), and other models (Argent, 2004). In most of these cases, GIS is used for loose coupling of the models, implementing data exchange, and visualization. GIS helps to overcome some of the problems related to data interoperability (Goodchild et al., 1997), particularly organizing data manipulation and passing data from one model to another model, which is a key requirement for integrated modeling to work (Argent, 2004). Moreover, availability of

flexible scripting languages (i.e. Python) in modern GIS packages allows to develop interactive user interfaces within GIS under which the models can be linked (Tao et al., 1996).

Python as a model integration language

Python is an open source object-oriented programming language balancing high-level programming with low-level optimization (Aruoba and Fernandez-Villaverde, 2015). Though Python programs usually run slower than FORTRAN, Java, or C/C++ programs, Python has become popular due to its simpler syntax and less requirements on specialized knowledge about operation system and memory management (Schmitz et al., 2009).

From the model integration perspective, Python has specific libraries supporting scientific programming (SciPy), modeling and data analysis (Pandas), as well as visualizations and parallel computing (IPython). Another advantage is its language interoperability often used to glue other programming languages. Thus, Python has libraries supporting function calls from MatLab (MLabWrap), R (RPy), Excel (OpenPyxl), FORTRAN (F2PY, PyFort), Delphi (Python4Delphi), Java (Jyton, JPype, Jepp), Perl (PyPerl), PHP (PiP), C/C++ (Ctypes, Cython, SWIG), etc. Moreover, Python runs natively on Windows, Mac, and Linux operation systems. Thus, Python can facilitate interoperating modules implemented in other programming languages (Roberts et al., 2010) and has been successfully used to link such models (Schmitz et al., 2009).

Model wrappers

The ArcGIS Model Builder was used as a programming environment for Python wrappers linking various independent models and providing a user friendly interface. Using the ArcGIS Model Builder allows us to capitalize on the data management and visualization functionality of the ArcGIS package. In addition, the Python wrappers can be adapted to run also without ArcGIS. However, in that case the user may have to set some of the parameters and the file names in the Python code instead of using the graphical interface provided by the Model Builder.

As each model has specific running parameters, input and output files, in spite of having similar general structure, the wrapper was fine-tuned for individual models used. Table 1 presents the main components of the wrapper. An example of complete code for wrapping a model used in the case study is provided in Appendix 1 for more details.

The wrappers developed for independent models can be integrated in the ArcGIS Model Builder environment. The process flow diagram paradigm provided by the Model Builder is convenient to present the models' workflow and linkages.

The ESRI Arcpy library was used for getting and setting the model parameters as well as for displaying status messages in the ArcGIS geo-processing window. The os.system() standard function was used to call the actual models from the wrapper. In some cases, access to a model source code may be restricted or changing it is not desirable. To avoid any problems with the file paths used in the model, the default path was set to the folder containing the model using the os.chdir() command. The exchangeable output files were set as wrapper output parameters and can be used by other models. However, even if the models are comparable in terms of data being exchanged, it is rare that one model output file format identically matches the input file format of the other model (Droppo et al., 2010). Thus, if data manipulation is required before passing an output from one model to the second, the powerful data processing toolset of ArcGIS can be effectively used.

Table 1. The main components used in the wrappers.

Component	Code example

Initialization stage
Organizes the initialization of the model based on the provided parameters such as the path to the model program (executable or script), parameters file, the scenario name, simulation period, etc.
	# Get input arguments
	in_Program = arcpy.GetParameterAsText(0)
	in_Scenario = arcpy.GetParameterAsText(1)
	in_Year = arcpy.GetParameterAsText(2)

Running stage
Makes the folder containing the model executable as a default folder and runs the model from there
	# Run the model / program
	desc = arcpy.Describe(in_Program)
	sourceFilePath = desc.path
	os.chdir(sourceFilePath)
	os.system(in_Program)

Results output stage
Export output files (filename1, filename2, etc.) from the model output folder (outfolder) into a specified folder from where other models can use them
	# Export output files
	Output1 = sourceFilePath+"\\outfolder\\filename1"
	Output2 = sourceFilePath+"\\outfolder\\filename2"
	Output3 = sourceFilePath+"\\outfolder\\filename3"
	arcpy.SetParameter(5, Output1)
	arcpy.SetParameter(6, Output2)
	arcpy.SetParameter(7, Output3)

Case study

Instead of only developing a theoretical approach of model integration, this paper also aims at testing the proposed model integration solution with a case study. For the Washington–Baltimore Metropolitan Area, diverse models were integrated. The resulting modeling suite is fully operational and was tested thoroughly to ensure that the proposed integration method matches the requirements outlined in ''Key requirements'' section.

Individual models

Five models were integrated within this modeling suite, covering transportation, land use, mobile emissions, building emissions, and land cover.

Maryland statewide transportation model (MSTM). The MSTM is an advanced trip-based model developed since 2007 by the Maryland State Highway Administration (FHWA, 2014). It was designed to estimate the impacts of transportation investments, changes to land use development, and impacts from factors beyond state boundaries, particularly for freight.

The model input data include population and employment by model zones, highway and public transit networks, and data on travel behavior. The model outputs report traffic volumes and speed for the overall system, corridors, or individual links.

Simple integrated land use orchestrator (SILO). Initially developed for Minneapolis/St. Paul ([Moeckel, forthcoming), SILO has been implemented for the state of Maryland. It micro-simulates household relocation, demographic changes, and developers who add, upgrade, or demolish dwellings. Every household, person, and dwelling is treated as an individual object. SILO is designed as a discrete choice model. Spatial decisions, such as relocation,

development of new dwellings, etc., are modeled with Logit models (McFadden, 1978). Other decisions, such as getting married, giving birth to a child, etc., are modeled by Markov models that apply transition probabilities.

SILO uses the Public Use Microdata Sample to create individual households and their dwellings. The MSTM provides the zone-to-zone travel times by auto and public transit. SILO generates a synthetic population with households, persons, dwellings, and jobs for the base year 2000 and incrementally updates these dataset in one-year increments through 2040. Every year the MSTM runs, SILO provides updated sociodemographic data.

Mobile emissions model (MEM). The MEM estimates transportation emissions by applying emission of the MOVES2010 EPA[1] model to MSTM-generated traffic flows (Welch, 2013).

The MEM input data include road network, vehicle trips, temperatures by month and hour for each county in the study area, humidity, average speed distribution, the vehicle miles of travel (VMT) on varied road types, fuel formulation, and supply. MEM runs every time the MSTM has run. Running emissions are calculated by applying emissions factors per mile to model VMT for each link. Aggregate link-level emissions are also calculated by functional class and pollutant. Nonrunning emissions are calculated by applying emissions factors per vehicle to the precalculated vehicle population (Welch, 2013).

Building energy consumption and emissions model (BEM). The BEM estimates CO_2 emissions and energy consumption from the built environment within Maryland (Welch, 2013). It uses the building, location, and climate variables of each property to determine whether the structure is likely to combust fossil fuels on site. If the probability is greater than 50%, then the model calculates CO_2 emissions from local combustion based on a set of related multipliers derived from a regression of the data from the US Energy Information Administration's residential[2] and commercial building[3] energy consumption surveys.

SILO provides the building stock for BEM. CO_2 emissions and energy consumption are the primary outputs of the model.

Chesapeake Bay land change model (CBLCM). The CBLCM was developed by the United States Geological Survey (USGS) within the Chesapeake Bay Program. It uses a stochastic methodology to emulate residential urban land use development in Maryland over a series of predefined time segments. It is as an independent cellular automata model that translates exogenous county-level projections of population and employment to estimates of urban land demand and then spatially allocates that onto 30 m resolution raster cells. The locations of future growth are informed by data on protected lands, zoning, slopes, land cover, proximity to urban centers, and proximity to locations of recent job and housing growth.

The model calculates a probability surface for growth locations and allocates households and dwellings provided by SILO to raster cells. CBLCM generates fine-grained patterns of residential urban growth across the study area.

Data exchange

One- or two-way data flows are implemented between the described models. For example, MSTM provides travel times and auto-operating costs to SILO and number of trips and average speed distribution to MEM; SILO provides population, employment, and auto availability data to MSTM, building data to BEM and population, employment, and accessibility to CBLCM (Figure 1). Currently, the MEM and BEM models are just users of output data from MSTM and SILO, and their output is not used by any other models.

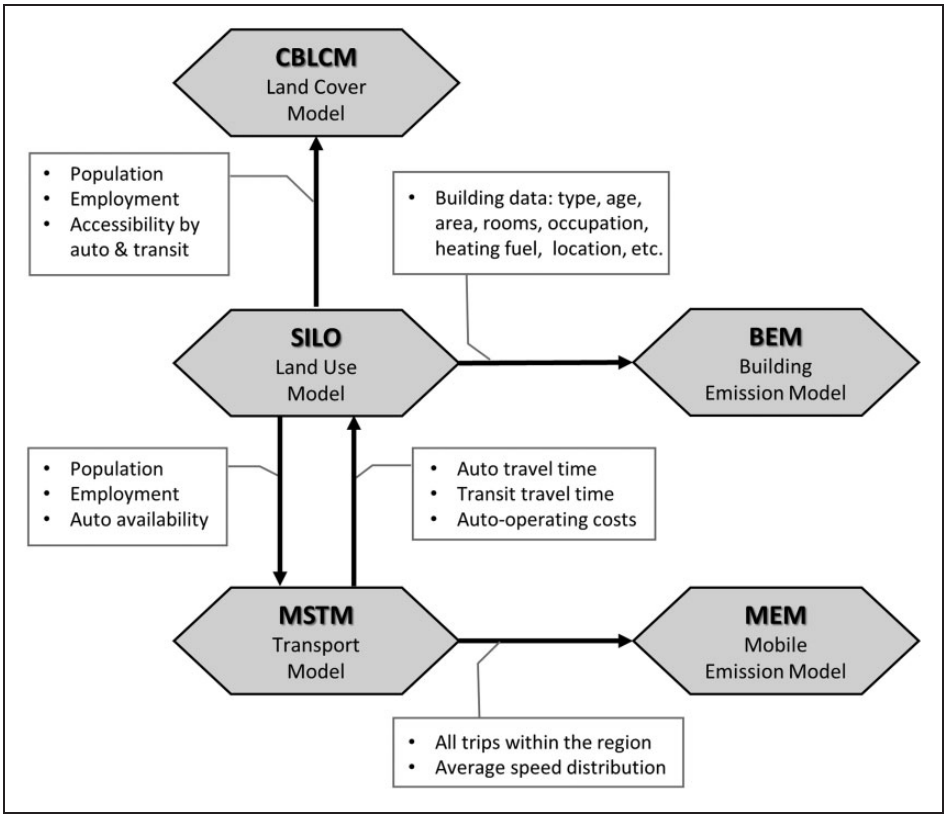

Figure 1. Data flow between the models.

Though the output from CBLCM is not used by the models involved in this study, it is used at USGS to feed a water quality model. In addition, there are many variables resulting from those models, which can be potentially used by other models.

This data exchange may be rather complex. As an example, the data exchange between the transportation model MSTM and the land use model SILO is visualized in Figure 2. Transit travel times are one of the factors that affect auto-ownership in SILO. The number of autos available in a household, on the other hand, affects the number of trips generated in the MSTM. The distribution of households and employment defines the trip origins and destinations in the two MSTM modules trip generation and trip distribution. The location choice of households, in turn, is affected by transit accessibility, auto-operating costs, and highway travel times. There are many interactions between the two models MSTM and SILO. These interactions are bidirectional and happen frequently throughout a model run.

At present, data exchange between those models is rather slow because the output from one model is written to a hard drive, that model shuts down, the other model starts and reads the data from the hard drive. Licensing and software limitations of the MSTM prohibited a closer integration with SILO. In some cases, this form of data exchange is time consuming and limits the intervals of data exchange to few simulation periods only. Though technically those data exchanges could be done for every simulation year, presently it is implemented only for selected simulation years (Figure 3). For example, population and employment data on 2030 from SILO are used by MSTM, BEM, and CBLCM simulations for 2030.

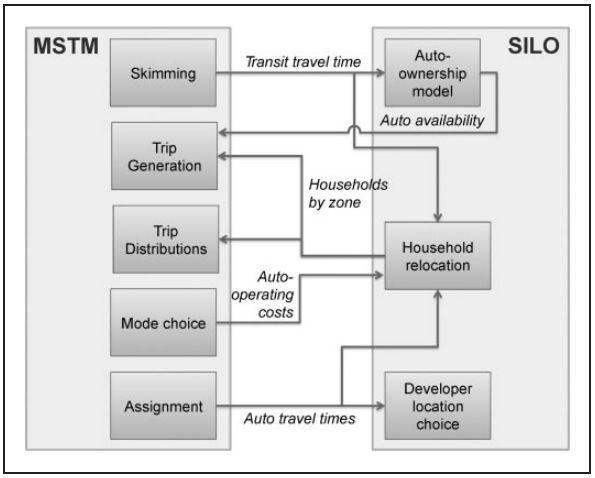

Figure 2. Data exchange between MSTM and SILO.

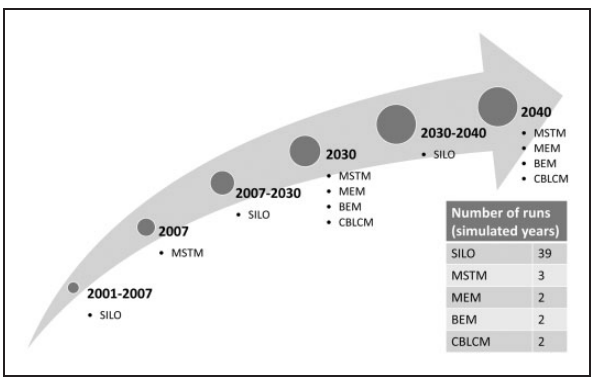

Figure 3. The models' processing flow order and simulation periods.

Then auto and transit travel times are obtained from the MSTM for the year 2030 and used as constant values in SILO for the years 2030–2040, while MSTM trip numbers and speed distribution are used by MEM to simulate mobile emissions for 2030. In the opposite direction, SILO writes out population and employment for 2040 and feeds those back to MSTM, BEM, and CBLCM in 2040. While SILO runs internally in one-year increments, MSTM, MEM, BEM, and CBLCM are run only for a few specific time points. Figure 3 presents the usual flow of the models. BEM, MEM, and CBLCM run only for future scenarios, while SILO and MSTM run also for historical datasets. Obviously, this can be modified based on the requirements of the particular research.

More frequent (e.g. annual) data exchange between SILO and MSTM is prohibited by the long model runtime of the MSTM. Running the original detailed version of the MSTM for one model year would take about 16 h. The simplified version of the MSTM applied in this study runs about 3 h for each model year (Table 2). And even with this simplified version it takes about 23 h to run the entire modeling suite. The implementation of the full feedback

Table 2. Main characteristics of the used models.

Model	Environment	Operation system	Developer/ licensing	Simulation years/period	Number of runs	Time per model year	Total runtime (h)[a]
SILO	Java	Multiplatform	Open source	2001–2040	39	9 min	6
MSTM	CUBE	Windows	Scripts: Open CUBE: CitiLabs	2007, 2030, 2040	3	3 h	9
MEM	CUBE	Windows	Scripts: Open CUBE: CitiLabs	2030, 2040	2	30 min	1
BEM	R	Multiplatform	Open source	2030, 2040	2	30 min	1
CBLCM	C / C++	CentOS Windows	USGS	2030, 2040	2	3 h	6

[a]20 x AMD Opteron Processor 6328 @ 3.20 GHz, 42 GB RAM, Windows 7.

between the models may deserve tighter integration. However, in addition to the difficulties with modifying model sources codes, language interoperability, and licensing restrictions, tighter integration may also result in a loss of performance measures such as speed, accuracy, or stability (Peckham et al., 2013). As a result, limited data exchange frequency is weighted against the advantage of supporting different types of components and linking them under a single user interface without changing their source codes. In this work, models developed in Java, CUBE script, R, and C++ have been coupled, while the automated coupling with an Excel-based model is under consideration.

Practical value of integrated modeling

Integrating various models helps to better represent complex interactions observed empirically between land use, transportation, and environment in the Baltimore–Washington region (Figure 4). Multiple agencies at the US federal, state, and local level have an interest in linking such models. This includes the US Environmental Protection Agency and the Maryland Departments of Transportation, Environment, Natural Resources and Planning. From a scientific perspective, this work aims to improve our understanding of human activity and environmental linkages, and to enable improved policy analysis dealing with environmental sustainability.

Model integration results

As described above, the models are developed in different programming environments in this application; they have various licensing requirements and run sequentially, which makes loose coupling method the most appropriate approach. The existing couplers could not been applied for various reasons, including the absence of support for the Windows operation system or a specific programming language as well as the requirements for changing the original codes. Therefore, the Python wrappers approach described above has been found to be superior for this type of integration task. The structures of the wrappers for each model are similar, with a different number of input and output files and parameter sets. Appendix 1 shows a sample Python code used to run the transportation model from ArcGIS. Here, Cube2omx[4] simple matrix converter is used within python wrapper to convert Citilabs Cube skims resulting from MSTM to OMX format required by SILO.

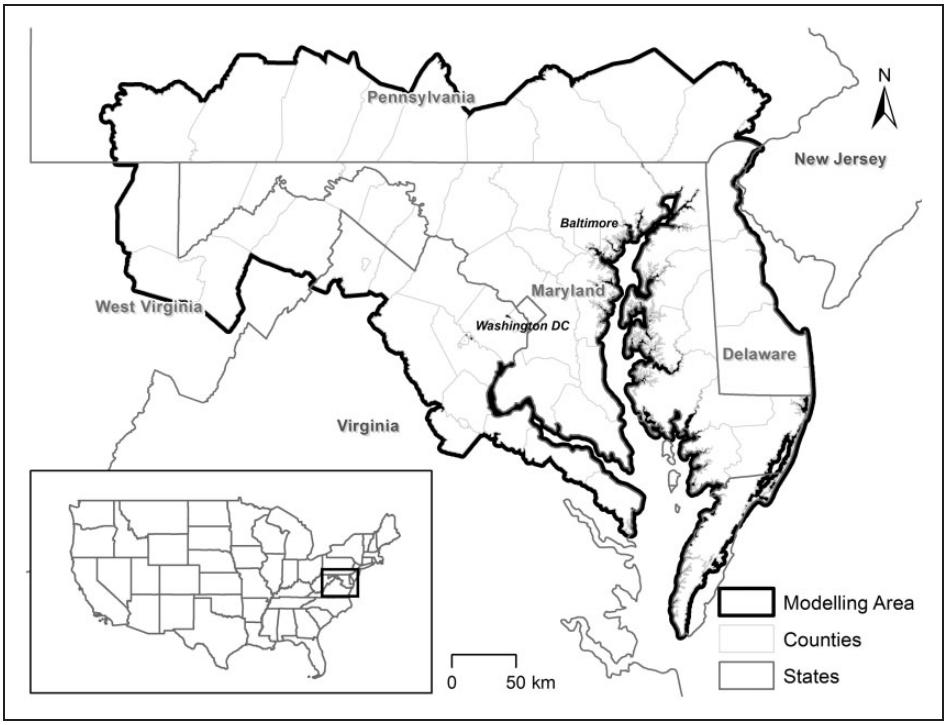

Figure 4. Baltimore–Washington region.

The geo-processing workflow model developed in ArcGIS Model Builder consists of a set of coupled modules (Figure 5). Solid lines represent data flow directions between models, while the dotted lines represent the preconditions to run the modules. The direction of the links and the preconditions determine a suitable order for model execution. Execution begins with models that have no incoming links or preconditions and proceeds to models whose preconditions have already been satisfied. All module parameters have their default values. However, they can be changed by double clicking on the relevant module/parameter icon. The wrapper codes can be viewed or edited by the ''Edit'' function available in the context menu, which opens the script in a text editor. The actual model code is not accessible from here, though their file paths are defined in the wrapper and can be used to open the models in their specific environment (e.g. CUBE, Microsoft Visual Studio, R Studio).

Models are executed for a specific simulation year or period defined by the model's parameters. The same model can be included in the geo-processing model multiple times to represent different simulation periods. Thus, the usual scenario simulation for this study starts with MSTM for the year 2012. The outputs of MSTM are then passed to SILO to model land use for the period of 2012–2030. As presented in Figure 5 SILO 2030 run is followed by MSTM, MEM, BEM, and CBLCM, which are using the output of SILO for 2030 and model results for year 2030. Though it is not presented in the figure, MSTM 2030 results are fed back to SILO to run it for the period of 2030–2040, which then again can be followed by MSTM, MEM, BEM, and CBLCM for 2040. As indicated previously, the output files of a model often need to be manipulated before passing them to the next model. This is particularly the case between the SILO and CBLCM models. SILO provides the household and employment numbers by zones, but CBLCM demands those

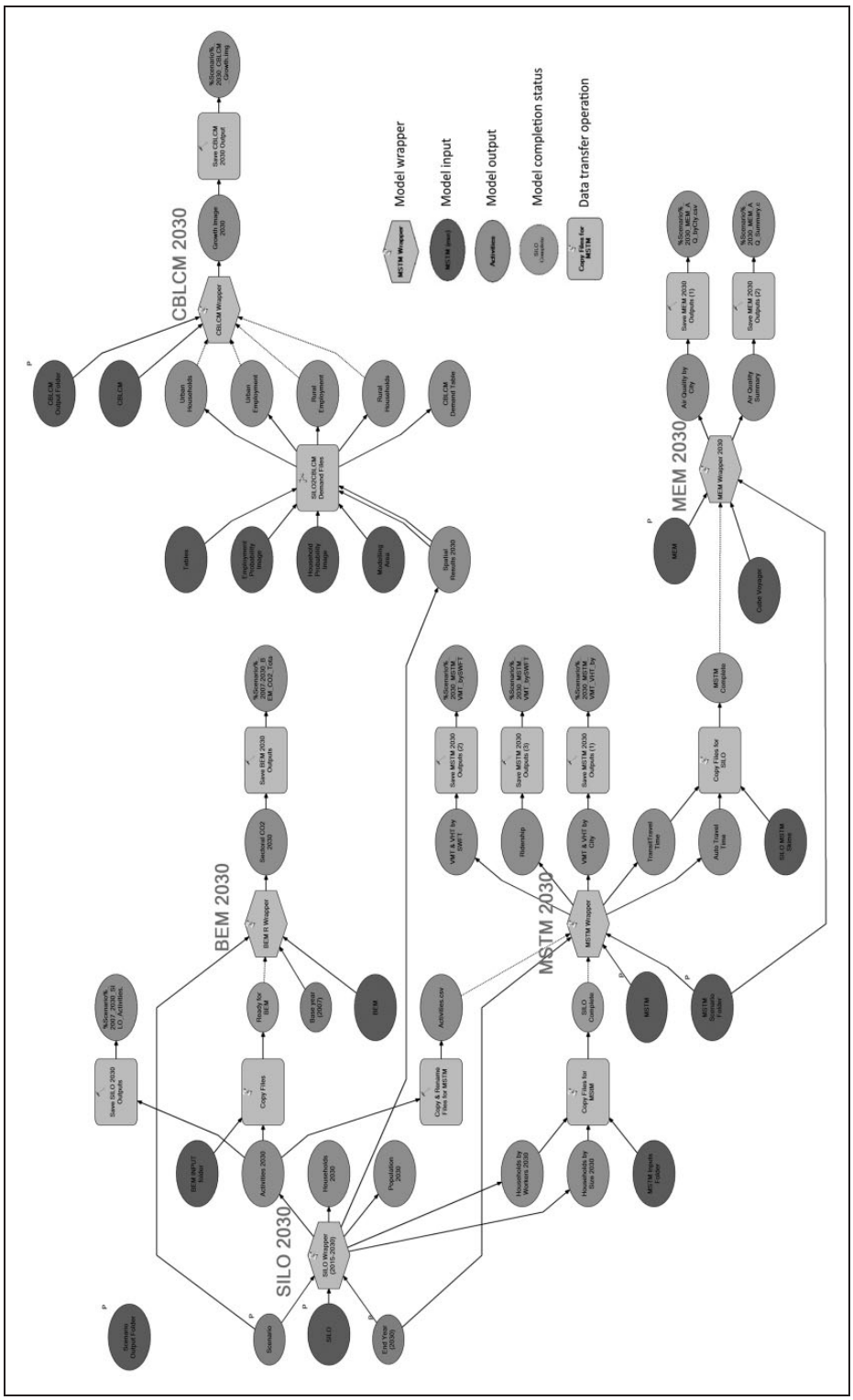

Figure 5. ArcGIS geo-processing model organizing the models simulation workflow.

numbers at the county level and distributed to urban/rural areas. To implement this efficiently, a special tool (SILO2CBLCM) has been developed and included in the integration suite (Figure 5). This tool apply existing functionality of ArcGIS (Reclassification, Tabulate Area, Join, Calculate Field, Query, etc.) as well as specifically developed Python scripts for table operations (transposing, exporting as csv files, etc.). Data transfer between other models is straightforward and only required copying the output files into designated input folders of subsequent models.

In addition to a standard status log by the ArcGIS geo-processing window, component models have their separate log files delivering detailed information on the specifics of the model run.

The implementation of integrated models was tested and compared with their stand-alone applications for the Washington–Baltimore region. As expected, the model outputs are almost identical. However, the following practical advantages of the integrated run have been proved during the case study.

Required efforts and implementation time. Each model has its own interface. They run through batch files that call the executable or script of the model and provide simulation years, scenario name, and the name of a parameter file. To run all the models independently, the user has to do the relevant changes manually for all models for every scenario and simulation year. This is an error-prone process. For example, if the user missed to change the simulation year or the scenario name in a model, the model would still run and provide output using erroneous settings, which then would be used by other models generating arbitrary results. The integrated suite, by contrast, provides a user-friendly interface and single entry for selected parameters, such as scenario name and simulation years, which then are used consistently by all models. When running the models independently, the user has to wait for a model to complete and then has to find and copy as well as possibly adjust, format, and rename the relevant files to be able to use them as input for subsequent models. This is time consuming and also prone to user errors. A small inaccuracy in formatting or a typo in a file name may cause model failure, or even worse, an error that is never discovered. The integrated suite implements those tasks automatically. Moreover, the user does not need to watch until a model run is completed, as the suite will launch subsequent models automatically whenever the proceeding model run completed. Taking into account that some of the model runtimes are several hours, this allows the user to run the complete model suite overnight, while in case of stand-alone runs the user has to check the model completion periodically to prepare and launch the run of the next model. While the run time of each single model is the same for stand-alone and integrated runs, the integrated suite may save hours in running all required models because of this automated integration.

Managing model outputs. In case of stand-alone runs, each model saves its output using the default folder and file names specified by the parameter files or model developers. In other words, the user has to be familiar with the folder structure and default naming used in all models and obtain the output files manually. For the integrated run, the relevant output files of all models can be renamed automatically to easily identifiable names (including the model name, scenario abbreviation, and simulation period) and copied into a user-specified folder. In addition, based on the output files, summary files with descriptive statistics are also created to report main indicators for each scenario.

Easiness to use. Running multiple models independently and using manual data transfer methods to couple them require detailed understanding of all involved models. The user

should know the sequence of the models' execution, the files required to exchange, and the paths of input/output folders for each of the model. While a detailed knowledge of all models involved is beneficial, complex integration tasks often require to include models of highly different domains, making it less likely that one experts understands each model in detail. With an increased number of models, this task becomes more and more complicated. As a result, only a limited number of expert users usually are able to run and follow the correct implementation of the complete set of models. In such cases, designing a user-friendly and intuitive user interface is a critical implementation task. A framework with minimal installation and configuration requirements will have higher usability (Belete et al., 2014). For this integrated suite, ArcGIS Model Builder provides a user-friendly interface that makes the links between the models obvious and that keeps track of the files required to be exchanged as well as the sequence of the model runs. The default values and help tips included in each tool window make its usable even for users who are not familiar with all the models and their requirements. The user is not even required to know the sequence of the implementation and just should provide the basic parameters, such as the simulation year, scenario name, script locations, and so on. These settings are for most part explained in the help file of the suite. Using the batch run functionality of ArcGIS Model Builder, it is also possible to setup and run multiple scenarios consecutively, without interim user input. Finally, all wrappers and data manipulation tools are included in a single ArcGIS Toolset package (available in GitHub), which only requires the standard installation and setup for ArcGIS.

Discussion

Full tight integration based on rewriting all the model codes in a single programming language and keeping all the data in memory is often considered to be the golden standard for model integration. However, in addition to being expensive and time consuming, this approach freezes the involved models in their current status, and makes it less likely to implement further updates done by the original developers at a later point in time. Such modeling suites also often become quite complex and difficult to apply (Beven, 2007). Moreover, the addition of any other model to the suite often requires extensive changes to the software, as the new model would need to be rewritten in the same language, and the data exchange mechanism would need to be adjusted, too. This makes the integrated system less flexible and harder to enrich with new models. The application of model coupling tools and frameworks, though at a smaller scale, also requires model code modifications and is potentially labor intensive. A typical development requires assimilating dissimilar components created by different developers and a limiting factor can be the compatibility of the spatial and temporal design of the framework with the external components (Droppo et al., 2010). Thus, PCRaster requires that the runtime of the component models must be separable into timescales given by the framework; and Schmitz et al. (2009) highlights that this is only practicable for systems with few components. For a large number of components, integration frameworks such as OpenMI or CSDMS are more efficient. However, they require the modification of the source codes and support only specific programming languages (Belete et al., 2014; Whelan et al., 2014). Even using generic model wrappers provided by FRAMES can be quite labor intensive (Droppo et al., 2010).

Most existing works in urban and regional integrated modeling include only a small number of components, such as land use and transportation models, occasionally linked

with demographic or economic models (Waddell, 2002; White et al., 2012) or transportation models integrated with emission models (Kaddoura and Nagel, 2016; Osorio and Nanduri, 2014). And in most of these cases the models have been rewritten into a single software package (e.g. UrbanSim, Metronamica, MOLAND, MATSim), or partially modified to be used by a coupling toolkit (e.g. BASINS, SEAMLESS), or the coupling is based on manual data transfer. The presented approach is a rare example of integrating a larger number of independent models without changing the source codes and providing a user-friendly interface for practical applications. Moreover, methods of cascading data from model to model often limit data flow in one direction only (Karcher et al., 2012). The described approach overcomes this limitation via stopping a model and resuming it after exchanging data with other models (e.g. data exchange between SILO and MSTM). Also, without a master data repository existing loose coupling approaches usually have limitations in tracking or documenting the inputs and outputs of an integrated model simulation (Karcher et al., 2012). By contrast, the above-described integration suite automatically gathers all model outputs into a user-specified folder making it easy to track and find required output. As it is integrated with ArcGIS, there is also an option to combine all these output files into a single geodatabase and use map templates for data visualization. Finally, some of the existing relevant frameworks work mainly with raster-based models (e.g. PCRaster) (Schmitz et al., 2009) or individual- or agent-based models (e.g. NetLogo) (Hjorth et al., 2015). In contrast, this approach can link an individual-based model with a raster-based model (e.g. coupling SILO and CBLCM).

With the use of Python wrappers, the implementation of the coupler is separated from the models' source codes. The loosely coupled geo-processing model in ArcGIS Model Builder provides a user-friendly interface and uses a data flow diagram paradigm to organize models and information flow. This provides plenty of flexibility, which helps in terms of portability, performance, and maintenance of the codes. While there are limitations, this approach supports different types of components and links them under a single user interface without changing their original source codes. The integrated system of the case study automatically calls component models developed in Java (SILO), CUBE script (MSTM and MEM), C++ (CBLCM), and R (BEM) environments (Figure 5). The presented approach allows adding a new model quite easily, requiring major revisions neither of the existing suite nor the model to be added. For example, it is intended to add the Integrated Transport and Health Impact Modeling Tool available in MS Excel and R environments (Woodcock et al., 2014).

Key benefits and limitations of the system are summarized below.

Benefits

- Open source;
- No need to change the source codes of the models;
- Allows to run models developed in different programming languages and file formats (e.g. exe, dll, bat, jar);
- Can be extended with additional models over time;
- User interface to show process flows and linkages between the models;
- Easy tracking of model input and output files;
- Availability of wide documentation and support on Python and ArcGIS;
- Rich visualization, spatial analytic and mapping capabilities through integration with ArcGIS;

- Easy to implement with regard to required time, resources, and programming experience.

Limitations

- Parallel model runs and dynamic data exchange during simulation time steps are not supported;
- Model processes run independently from one another;
- Data exchanged between modules need to be written to and read from a hard drive. No in-memory data exchange is available;
- Not needing to know the used models in detail is a benefit, but also carries the risk that models are applied in ways and for purposes not adequate for the model.

The study presents one of the first attempts to couple as diverse models without changing their source codes. Specifically, the method was applied in the case study to couple a microsimulation discrete-choice land use model, a cellular-automata-based land cover model, a trip-based transportation model, a factor-based mobile and a logit-based building emission models. However, the methodology is not limited to this type of systems. It can also be applied to other systems requiring consecutive implementation of stand-alone components including nonspatial models. The Python wrappers and ArcGIS models are publicly available at GitHub[5] and can be used as templates for the integration of other models.

Conclusion

Close model integration has become the mantra among model developers. Integration tools under development, such as OpenMI or CSDMS, promote tight integration of different models and ease information transfer between the same. Continuously increasing computational capacities enables ever more comprehensive model integrations. From a technical perspective, the prospects of tight model integration are excellent.

However, the research presented here also exemplified limitations and challenges of model integration. Attempts to tightly integrate specific models using existing couplers and integration frameworks failed due to a lack of software compatibility and licensing restrictions. Instead, a less tight but robust Python/GIS-based data exchange integration has been implemented, completely satisfying the research goals. Notwithstanding the ability to run complex model scenarios, the probably most important lesson learned of this research refers to the level of model integration. While the initial attempt was to integrate all models as tightly as possible, the authors developed a much more agile integration approach. An important finding of this research is that model integration methods should depend on direction of information exchange and frequency of data flows.

Model direction refers to the sender model and receiver model of information. For example, an economic model is used to provide regional control totals of population and employment growth for the entire study area. While the land use model allocates this growth to individual zones, the overall growth is provided exogenously by this national economic input/output model. In theory, the performance of this study area could be fed back into the economic model, as for example tighter land use restrictions could push some growth to neighboring regions. In reality, however, the impacts of scenarios for the Baltimore–Washington area on the national economy are minimal. Raising tolls or restricting land

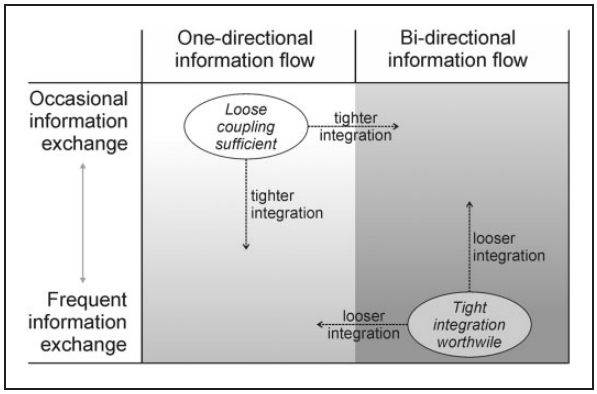

Figure 6. Reasons for loose coupling and tight integration.

development is unlikely to have a notable effect of growth in this study area. Given that economic growth is used as a one-way flow of information, the integration between the economic model and the land use model is kept offline and solved with a single file transfer covering a 40-year growth forecast. This model linkage is represented by the oval ''Loose coupling sufficient'' in Figure 6.

The second aspect of model integration is the frequency of interaction. For example, mobile emissions are calculated every time after the transportation model ran. This is a one-way flow of information: transportation generates emissions and emissions (commonly) do not affect the travel behavior. However, even though this is a one-way flow of information, the exchange of information is frequent enough that the transportation model and the mobile emission model warrant closer integration. Frequent data flows deserve closer integration to ease information flow, even if the flow is only happening in one-way direction (lower left quadrant of Figure 6).

The tightest integration should be pursued for models that exchange information frequently and bidirectionally. In the modeling suite presented in this paper, this level of integration applies to the land use and transportation models. These models exchange information in both directions: the location of households and employment define the origins and destinations in the transportation model, and travel times are converted into accessibilities that affect household relocation decisions. Given the frequency of this bidirectional flow, these two models deserve most attention for data exchange methods to ensure information exchange with little translation loss and limited impact on model runtime.

More integration is not always better. Using the appropriate level of integration improves model stability and runtimes without compromising important linkages between models.

Acknowledgement

The research presented in this paper benefited from many discussions with Uri Avin, Frederick Ducca, Daniel Engelberg, Sevgi Erdogan, Gerrit Knaap, Timothy Welch, Peter Claggett, and Di Yang.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported by a Marie Curie International Outgoing Fellowship (GeoSInPo) within the 7th European Community Framework Program and by the National Socio-Environmental Synthesis Center (SESYNC) under funding received from the US National Science Foundation DBI-1052875.

Notes

1. http://www.epa.gov/otaq/models/moves
2. http://www.eia.gov/consumption/residential
3. http://www.eia.gov/consumption/commercial
4. https://github.com/osPlanning/cube2omx
5. https://github.com/Shahumyan/ModelPythonWrappers

References

Argent RM (2004) An overview of model integration for environmental application – Components, frameworks and semantics. *Environmental Modelling and Software* 19: 219–234.

Aruoba SB and Fernandez-Villaverde J (2015) A comparison of programming languages in macroeconomics. *Journal of Economic Dynamics and Control* 58: 265–273.

Babendreier JE and Castleton KJ (2005) Investigating uncertainty and sensitivity in integrated, multimedia environmental models: Tools for FRAMES-3MRA. *Environmental Modelling and Software* 20: 1043–1055.

Basnyat P, Teeter LD, Lockaby BG, et al. (2000) The use of remote sensing and GIS in watershed level analyses of non-point source pollution problems. *Forest Ecology and Management* 128: 65–73.

Belete GF, Voinov A and Holst N (2014) An architecture for integration of multidisciplinary models. In: *Seventh Intl. Congress on Env. Modelling and Software*. San Diego, CA: International Environmental Modelling and Software Society (iEMSs), pp. 1251–1259.

Berry J and Buckley DTR (1997) Seamlessly linking ARC/INFO to landscape analysis and forest growth models. In: *Proceedings of Forest Vegetation Simulator Conference*, Fort Collins, CO, USA, pp. 21–29.

Beven K (2007) Towards integrated environmental models of everywhere: Uncertainty, data and modelling as a learning process. *Hydrology and Earth System Sciences* 11: 460–467.

Brandmeyer JE and Karimi HA (2000) Coupling methodologies for environmental models. *Environmental Modelling and Software* 15: 479–488.

Brazier RE, Heathwaite AL and Liu S (2005) Scaling issues relating to phosphorus transfer from land to water in agricultural catchments. *Journal of Hydrology* 301: 330–342.

Clarke KC and Gaydos LJ (1998) Loose-coupling a cellular automaton model and GIS: Long-term urban growth prediction for San Francisco and Washington/Baltimore. *International Journal of Geographical Information Science* 12: 699–714.

Devantier BA and Feldman AD (1993) Review of GIS applications in hydrologic modeling. *Journal of Water Resources Planning and Management—ASCE* 119: 246–261.

Droppo JG, Whelan G, Tryby ME, et al. (2010) Methods to register models and input/output parameters for integrated modeling. In: *International Congress on Environmental Modelling and Software*. Ottawa, Canada: International Environmental Modelling and Software Society (iEMSs). Available at: http://www.iemss.org/iemss2010/papers/S10/S.10.21.Methods%20to%20Register%20Models%20and%20InputOutput%20Parameters%20for%20Integrated%20Modeling%20Frameworks%20-%20GENE%20WHELAN.pdf

FHWA (2014) *Maryland State Highway Administration Maryland Statewide Travel Model (MSTM) Peer Review Report*. Washington, DC: US Department of Transportation Federal Highway Administration. Available at: https://www.fhwa.dot.gov/planning/tmip/resources/peer_review_program/maryland_statewide/mdsha.pdf

Geertman S and Stillwell J (2009) Planning support systems: Content, issues and trends. *Planning Support Systems Best Practice and New Methods* 95: 1–26.

Goodchild MF, Egenhofer MJ and Fegeas R (1997) Interoperating GISs. In: *Report of a specialist meeting held under the auspices of the Varenius ProjectNCGIA,* Santa Barbara, CA, USA: National Center for Geographic Information and Analysis, p.64.

Gregersen JB, Gijsbers PJA and Westen SJP (2007) OpenMI: Open modelling interface. *Journal of Hydroinformatics* 9: 175–191.

Hill C, DeLuca C, Balaji, et al. (2004) The architecture of the earth system modeling framework. *Computing in Science and Engineering* 6: 18–28.

Hjorth A, Brady C, Head B, et al. (2015) Thinking within and between levels: Exploring reasoning with multi-level linked models. In: Koschmann PHT and Tchounikine P (eds) *Proceedings of the Computer Supported Collaborative Learning (CSCL) Conference*. Gothenburg, Sweden: ISLS.

Kaddoura I and Nagel K (forthcoming) Activity-based computation of marginal noise exposure costs: Impacts for traffic management. *Transportation Research Record* 2597. DOI: 10.3141/2597-15.

Karcher SC, VanBriesen JM and Nietch CT (2012) Assessing the challenges associated with developing an integrated modeling approach for predicting and managing water quality and quantity from the watershed through the drinking water treatment system. National Risk Management Research Laboratory, U.S. Environmental Protection Agency, EPA/600/R-12/030.

Kittle JL, Duda PB, Ames DP, et al. (2006) The BASINS watershed analysis system – Integrating with open source GIS. In: *Geographic Information Systems and Water Resources IV – AWRA Spring Specialty Conference*. Houston, TX: AWRA.

Lam D, Leon L, Hamilton S, et al. (2004) Multi-model integration in a decision support system: A technical user interface approach for watershed and lake management scenarios. *Environmental Modelling and Software* 19: 317–324.

Laniak GF, Olchin G, Goodall J, et al. (2013) Integrated environmental modeling: A vision and roadmap for the future. *Environmental Modelling and Software* 39: 3–23.

León LF, Soulis ED, Kouwen N, et al. (2002) Modeling diffuse pollution with a distributed approach. *Water Science and Technology* 45: 149–156.

Liu S, Brazier RE, Heathwaite AL, et al. (2014) Fully integrated approach: An alternative solution of coupling a GIS and diffuse pollution models. *Frontiers of Environmental Science and Engineering* 8: 616–623.

McFadden D (1978) Modelling the choice of residential location. In: Anders K, Lars L, Folke S, et al. (eds) *Spatial Interaction Theory and Planning Models*. Amsterdam, New York, Oxford: North-Holland Publishing Company, pp. 75–96.

Moeckel R (forthcoming) Constraints in household relocation: Modeling land-use/transport interactions that respect time and monetary budgets. *The Journal of Transport and Land Use* 10(2). Available at: https://www.jtlu.org/index.php/jtlu/article/view/810/723

Moeckel R, Mishra S, Ducca F and Weidner T (2015) Modeling complex Megaregion systems: Horizontal and Vertical Integration for a Megaregion Model. *International Journal of Transportation* 3(1): 69–90.

Moore RV and Tindall CI (2005) An overview of the open modelling interface and environment (the OpenMI). *Environmental Science and Policy* 8: 279–286.

Osorio C and Nanduri K (2014) Urban transportation emissions mitigation: Coupling high-resolution vehicular emissions and traffic models for traffic signal optimization. *Transportation Research Part B: Methodological* 81: 520–538.

Overeem I, Berlin MM and Syvitski JPM (2013) Strategies for integrated modeling: The community surface dynamics modeling system example. *Environmental Modelling and Software* 39: 314–321.

Peckham SD, Hutton EWH and Norris B (2013) A component-based approach to integrated modeling in the geosciences: The design of CSDMS. *Computers and Geosciences* 53: 3–12.

Piacentini A, Morel T, Thevenin A, et al. (2011) O-Palm: An open source dynamic parallel coupler. In: *IV International Conference on Computational Methods for Coupled Problems in Science and Engineering*, Kos Island, Greece. pp. 885–895.

Rahman JM, Cuddy SM and Watson FGR (2004) Tarsier and ICMS: Two approaches to framework development. *Mathematics and Computers in Simulation* 64: 339–350.

Roberts JJ, Best BD, Dunn DC, et al. (2010) Marine geospatial ecology tools: An integrated framework for ecological geoprocessing with ArcGIS, Python, R, MATLAB, and C plus. *Environmental Modelling and Software* 25: 1197–1207.

Schmitz O, Karssenberg D, van Deursen WPA, et al. (2009) Linking external components to a spatio-temporal modelling framework: Coupling MODFLOW and PCRaster. *Environmental Modelling and Software* 24: 1088–1099.

Shah AR, Castleton KJ and Hoopes BL (2004) Framework for risk analysis in multimedia environmental systems: Modeling individual steps of a risk analysis process. In: *Proceedings of the International Conference on Modeling, Simulation & Visualization Methods, MSV'04 and Proceedings of the International Conference on Algorithmic Mathematics & Computer Science, AMCS'04*, 21–24 June 2004, Las Vegas, Nevada, USA. CSREA Press.

Sudicky E, Vanderkwaak J, Jones J, et al. (2003) Fully-integrated modelling of surface and subsurface water flow and solute transport: Model overview and application. *Developments in Water Science* 50: 313–318.

Tao C, Kainz W and van Zuidam R (1996) Coupling GIS and environmental modelling: The implications for spatio-temporal data modelling. *International Archives of Photogrammertry and Remote Sensing* 31: 849–856.

Valcke S (2013) The OASIS3 coupler: A European climate modelling community software. *Geoscientific Model Development* 6: 373–388.

van Delden H, Seppelt R, White R, et al. (2011) A methodology for the design and development of integrated models for policy support. *Environmental Modelling and Software* 26: 266–279.

van Ittersum MK, Ewert F, Heckelei T, et al. (2008) Integrated assessment of agricultural systems – A component-based framework for the European Union (SEAMLESS). *Agricultural Systems* 96: 150–165.

van Walsum PEV and Veldhuizen AA (2011) Integration of models using shared state variables: Implementation in the regional hydrologic modelling system SIMGRO. *Journal of Hydrology* 409: 363–370.

Voinov A and Shugart HH (2013) 'Integronsters', integral and integrated modeling. *Environmental Modelling and Software* 39: 149–158.

Waddell P (2002) UrbanSim – Modeling urban development for land use, transportation, and environmental planning. *Journal of the American Planning Association* 68: 297–314.

Warner JC, Perlin N and Skyllingstad ED (2008) Using the model coupling toolkit to couple earth system models. *Environmental Modelling and Software* 23: 1240–1249.

Welch TF (2013) *Climate Action Plans – Fact or Fiction? Evidence from Maryland*. College Park: Faculty of the Graduate School, University of Maryland.

Whelan G, Kim K, Pelton MA, et al. (2014) Design of a component-based integrated environmental modeling framework. *Environmental Modelling and Software* 55: 1–24.

White R, Uljee I and Engelen G (2012) Integrated modelling of population, employment and land-use change with a multiple activity-based variable grid cellular automaton. *International Journal of Geographical Information Science* 26: 1251–1280.

Wong I, Lam D, Booty W, et al. (2009) A loosely-coupled collaborative integrated environmental modelling framework. In: *Americas Conference on Information Systems*, AIS Electronic Library, Paper 741.

Woodcock J, Tainio M, Cheshire J, et al. (2014) Health effects of the London bicycle sharing system: Health impact modelling study. *British Medical Journal* 348: g425.

Harutyun Shahumyan is a Marie Curie Research Fellow from University College Dublin currently hosted at the HYPERLINK "http://smartgrowth.umd.edu/" \t "_self" National

Center for Smart Growth at the University of Maryland. His research is focused on geospatial modelling, model integration and application of spatial decision support systems for policy analysis. Harutyun holds a Ph.D. in Engineering from the National Academy of Sciences of Armenia and has several years of post-doctoral research experience in University College Dublin, where his work was focused on land use modelling, scenario development and geospatial analysis. Previously, he worked as a GIS and Data Management expert in leading international organizations such as USAID, Emerging Markets Group, PA Consulting and Development Alternatives Inc.

Rolf Moeckel is an Assistant Professor in the Department of Civil, Geo and Environmental Engineering at the Technical University of Munich in Germany. He leads a research group called Modeling Spatial Mobility that focuses on land use and transportation modeling. Previously, he worked as a PostDoc at the National Center for Smart Growth at the University of Maryland, where the research for this paper was conducted. Rolf developed the land use model SILO while working as a consultant with Parsons Brinckerhoff in New York and Albuquerque. He holds a doctorate in spatial planning from the University of Dortmund in Germany.

Appendix 1: Python wrapper for the transportation model

```
#*********************************************************************
  # MSTM model wrapper
  # Arguments:
  # 0 - MSTM Model executable file
  # 1 - MSTM Scenario folder
  # 2 - Modelling year (should be similar to recent SILO End year)
  # 3 - Output for SILO 1: HwyPK_iter6.omx (Saved in Scenario folder\Outputs)
  # 4 - Output for SILO 2: WTrnPK.omx (Saved in Scenario folder\Outputs)
  #*********************************************************************
  # Standard error handling
  try:
  import arcpy
  import time
  import os
  import string
  start = time.time()
  arcpy.AddMessage(“““)
  arcpy.AddMessage(“MSTM model start time: %s” % time.strftime(‘%X %x %Z’))
  # Get input arguments
  in_Program = arcpy.GetParameterAsText(0)
  in_Scenario = arcpy.GetParameterAsText(1)
  in_Year = arcpy.GetParameterAsText(2)
  # Check that the program exist
```

```
if not arcpy.Exists(in_Program):
raise Exception, "Input program does not exist"
# Run the model / program
arcpy.AddMessage("")
arcpy.AddMessage("Running %s" % (in_Program))
desc = arcpy.Describe(in_Program)
sourceFilePath = desc.path
os.chdir(sourceFilePath)
os.system(in_Program)
# Export shared files
sourceFilePath = in_Scenario
skm2SILO1=sourceFilePath+"\\HwyPK_iter6.skm"
skm2SILO2=sourceFilePath+"\\WTrnPK.skm"
# Convert Skim Matrices from the MSTM to OMX Matrices for SILO
cube2omx_converter = "C:\\models\\cube2omx\\cube2omx.exe"
os.system(cube2omx_converter+ " " + skm2SILO1)
os.system(cube2omx_converter+ " " + skm2SILO2)
omx2SILO1=sourceFilePath+"\\HwyPK_iter6.omx"
omx2SILO2=sourceFilePath+"\\WTrnPK.omx"
# Rename output files to include the modelling year as defined in SILO properties file
omx2SILO1_year=sourceFilePath+"\\HwyPK_iter6_"+str(in_Year)+".omx"
omx2SILO2_year=sourceFilePath+"\\WTrnPK_"+str(in_Year)+".omx"
if arcpy.Exists(omx2SILO1_year):
   os.remove(omx2SILO1_year)
os.rename(omx2SILO1, omx2SILO1_year)
if arcpy.Exists(omx2SILO2_year):
   os.remove(omx2SILO2_year)
os.rename(omx2SILO2, omx2SILO2_year)
arcpy.AddMessage("Exporting exchange data files:")
arcpy.AddMessage(omx2SILO1_year)
arcpy.AddMessage(omx2SILO2_year)
arcpy.SetParameter(3, omx2SILO1_year)
arcpy.SetParameter(4, omx2SILO2_year)
elapsed = (time.time() - start)
arcpy.AddMessage("")
arcpy.AddMessage("Model end time: %s" % time.strftime('%X %x %Z'))
# Handle script errors
except Exception, errMsg:
# If we have messages of severity error (2), we assume a GP tool raised it.
# Otherwise, we assume we raised the error and the information is in errMsg.
if arcpy.GetMessages(2):
arcpy.AddError(arcpy.GetMessages(2))
else:
arcpy.AddError(str(errMsg))
```