Ecological Informatics 32 (2016) 124–133

Contents lists available at ScienceDirect

Ecological Informatics

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

A software framework for process flow execution of stochastic multi-scale integrated models

Oliver Schmitz [a,b,*], Jean–Luc de Kok [b], Derek Karssenberg [a]

[a] Department of Physical Geography, Faculty of Geosciences, Utrecht University, Heidelberglaan 2, PO Box 80115, 3508 TC Utrecht, The Netherlands
[b] Flemish Institute for Technological Research (VITO), Unit Environmental Modelling, Boeretang 200, 2400 Mol, Belgium

ARTICLE INFO

Article history:
Received 12 October 2015
Received in revised form 25 January 2016
Accepted 27 January 2016
Available online 4 February 2016

Keywords:
Spatio-temporal modelling
Uncertainty assessment
Integrated modelling
PCRaster Python
Model components

ABSTRACT

Dynamic environmental models use a state transition function, external inputs and parameters to simulate the change of real-world processes over time. Modellers specify the state transition function and the external inputs required in the process calculation of each time step in a component model, a self-contained numerical module representing an individual spatio-temporal process. Depending on the application case of a component model – such as standalone execution or in an integrated model – the source of the external input needs to be specified. The required external inputs can thereby be obtained by a file operation in case of a standalone execution. Alternatively, required inputs can be obtained from other component models, in case the component model is part of an integrated model. Using different notations to specify these input requirements, however, requires a modification of the state transition function per application case and therefore would reduce the generic applicability of a component model.

To address this problem, we propose the function object notation as a means to specify the input requirements of a component model. This function object notation provides modellers with a uniform syntax to express the input requirements within the state transition function. During component initialisation, the function objects can be parameterised with different external sources. In addition to a uniform syntax, the function object notation allows a modeller to specify a request-reply execution flow of the coupled models (i.e. a component requests data needed for its own progress from another component). We extend the request-reply execution approach to Monte Carlo simulations and implement a software framework prototype. Using this prototype, we build an exemplary integrated model by coupling components for land use change, hydrology and Eucalyptus tree growth at different temporal discretisations to obtain the probability for bioenergy plant growing in a hypothetical catchment. The presented approach allows modellers to specify input requirements in the state transition function independently from the source of external inputs and therefore increases the reusability of these component models.

1. Introduction

Integrated environmental research faces the practical challenge of combining knowledge across different scientific disciplines such as hydrology, ecology, agricultural or economic sciences. Spatio-temporal models from these various disciplines may operate at different spatial and temporal discretisations, depending on the processes represented (see, e.g. Hamilton et al., 2015; Jakeman et al., 2006; Kelly (Letcher) et al., 2013). Coupling such models allows for analysis from an integrated perspective, therefore improving our understanding of interactions between subsystems (e.g. Claessens et al., 2009; Elag et al., 2011; Rotmans, 1990; Verburg, 2006). Moreover, such integrated models are important tools for environmental management and integrated assessment (e.g. Ewert et al., 2009; Muth and Bryden, 2013; van Ittersum et al., 2008; Welsh et al., 2013). To obtain reliable results from these interdisciplinary models and to appropriately represent complex environmental systems, modellers are faced with two major challenges: (1) the construction of component models and their couplings aimed at an appropriate representation is given by the specific system under study, and (2) a need for error analysis to assess the reliability of their models and the uncertainty in particular of forecasts required for managing the real systems.

We refer to component models as the main building blocks that environmental scientists construct and couple into integrated models. A component model is a self-contained numerical module holding an individual spatio-temporal process representation of an environmental phenomenon. To obtain reusable component models, the process implementations in the component model should be independent of a particular modelling context or other component models. A more

* Corresponding author at: Department of Physical Geography, Faculty of Geosciences, Utrecht University, Heidelberglaan 2, PO Box 80115, 3508 TC Utrecht, The Netherlands. Tel.: +31 30 253 9363; fax: +31 30 253 1145.
E-mail address: o.schmitz@uu.nl (O. Schmitz).

http://dx.doi.org/10.1016/j.ecoinf.2016.01.009
1574-9541/

straightforward coupling of component models can be realised by means of standardised interfaces for inputs and outputs (see also Argent, 2005; Hinkel, 2009; Rizzoli et al., 2008).

Model builders can choose from a variety of software applications supporting environmental modelling and analysis, including, amongst others, general purpose programming languages (e.g. Fortran, Java, C, Python), geographical information systems (e.g. ArcGIS, 2015; Neteler et al., 2012; QGIS Development Team, 2015), visual and domain specific modelling environments (e.g. ExtendSim, 2015; PCRaster, 2015; Pullar, 2004; Schwanghart and Kuhn, 2010; Schweitzer et al., 2011; Sklar, 2007), software frameworks supporting the development of integrated models (e.g. David et al., 2013; Hill et al., 2004; Hinkel, 2009; Moore and Tindall, 2005; Peckham et al., 2013), or a combination of several of these tools (e.g. Huang et al., 2015; Marohn et al., 2013; Roberts et al., 2010; Steiniger and Bocher, 2009). A modular development of reusable components as recommended amongst others by Villa and Costanza (2000), Argent (2005), Rizzoli et al. (2008), Holzworth et al. (2010), or de Kok et al. (2015) can be achieved by adopting software development principles such as object-oriented development or component-based software engineering practises (e.g. Booch et al., 2007; Gamma et al., 1995; Szyperski, 2002). However, the variety of available tools and expert programming knowledge required to apply these tools still pose a burden for domain specialists.

In addition to the complex construction process of integrated models, the numerical implementations are merely abstract interpretations of the real human–natural systems, and the resulting model predictions are subject to uncertainty. The main sources of model uncertainty are the conceptual uncertainty, such as the structure of a modelled system not being known or incompletely described, and the uncertainties of model inputs (system drivers and parameters), which are for example due to measurement errors or a lack of observational data (see Karssenberg and de Jong, 2005; Lindenschmidt et al., 2007; Refsgaard et al., 2006). Assessing the model output uncertainty is an essential task in environmental modelling, and recommended by several authors as part of good modelling practises (e.g. Beven and Binley, 1992; Jakeman et al., 2006; Matott et al., 2009; Refsgaard et al., 2007; Risbey et al., 2005). A common approach to assess model uncertainty is to express uncertainty using stochastic variables, and approximating the error in model outputs using Monte Carlo simulation. In Monte Carlo simulation, the model is run for a large number of realisations of model inputs to obtain probability distributions of the model state variables (see, e.g. Doucet et al., 2001; Heuvelink, 1998; Karssenberg and de Jong, 2005; Liu and Chen, 1998).

In summary, modellers need to construct component models representing individual environmental processes, they need to integrate component models to represent complex human–natural systems, and they need to analyse the integrated models. These tasks are done in an iterative development process (e.g. Jakeman et al., 2006; Jørgensen and Bendoricchio, 2001) by using different process representations and component compositions to test and evaluate underlying assumptions or different model structures (e.g. Del Giudice et al., 2015; Shen et al., 2014). Model component flexibility, i.e. the technical ease of use to apply a component in different modelling contexts, is essential for the plug-and-play construction, analysis and maintenance of integrated models (see, e.g. de Kok et al., 2015; Papajorgji, 2005). Modular development practises such as the component-based development practise (e.g. Szyperski, 2002) promote this flexibility by defining requirements related to the 1) the construction of confined components with defined input and output interfaces and 2) a communication protocol that is shared between all components. Modellers should preferably be able to follow these practises within one modelling environment at their own level of domain expertise (see also Karssenberg, 2002).

During component construction, the modeller specifies the input requirements of a component model. Depending on the usage of a component model, however, these requirements can be met during model runtime differently: if a component model is executed individually, the inputs can be obtained from precalculated data or observational data stored on disc. If a component model is part of an integrated model, the inputs are obtained from other component models at runtime. The software prototype introduced in (Schmitz et al., 2013) uses different operations to obtain inputs either from disc or from other component models. The component interfaces therefore depend on how input variables are obtained. This results in a technical dependency between the component interface and the process description. The resulting component models are therefore less generic related to software, although obtaining input variables either from disc or from other component models is conceptually equal. It is required to separate the dependency between process description and component interface to enhance the reusability of component models.

The objective of this paper is to develop a mechanism allowing domain specialists to uniformly describe process descriptions, and therefore the state transition function, independently of the origin of required component inputs. We provide this mechanism in one declarative development language enabling an environmental modeller to perform an exploratory model construction workflow while building and assessing integrated models (e.g. de Kok et al., 2015). To reach the objective, we first generalise the input requirements of a component model. Section 2 introduces a parameterisation of the component interface allowing for different input sources. We propose the function object notation as consistent syntax to describe input origins in the process descriptions. Based on this notation, we introduce the concept of function-based model execution, and outline the requirements for the joint execution of multi-scale spatio-temporal component models. Then, we extend the requirements to allow for uncertainty assessment based on Monte Carlo simulation. Based on these requirements, we describe in Section 3 a software framework prototype developed in the high-level scripting language (Python, 2015) using the PCRaster Python module (Karssenberg et al., 2010) to model spatio-temporal processes. The framework provides class templates as building blocks for integrated models that can be completed by environmental modellers with map algebra (Tomlin, 1990) operations. This is illustrated in a case study in Section 4 by applying the modelling framework for the construction, execution and visualisation of stochastic spatio-temporal component models. We conclude our paper with an evaluation of the presented approach.

2. Methodology

2.1. Specification of data requests

To construct integrated models, modellers need to be able to express specific dynamic processes over various spatial scales. These processes can be described according to a state transition function f transferring the previous state $Z(t_{i-1})$ using external inputs $I(t_i)$ and parameters P to the state of the current time step $Z(t_i)$ according to the following equation (Beck et al., 1993; Burrough, 1998):

$$Z(t_i) = f(Z(t_{i-1}), I(t_i), P) \quad \forall\ t_i \tag{1}$$

The state transition function f is represented in the numerical model by a set of predefined operations that allow for a flexible model construction process. These operations are calculated on spatial data types, provided for example by the map algebra concept and related implementations (e.g. Karssenberg et al., 2007; Schmitz et al., 2013; Tomlin, 1990). The origin of external inputs $I(t_i)$ need to be specified by the modellers as well. These inputs can be obtained, for example, by an operation reading a data set from disc, or they can be obtained as output from a different component model. The application context of the component model can require to obtain inputs from different data sources. When executing a component model individually, required inputs need to be obtained by reading data sets from disc storage.

When executing an integrated model, the required inputs can be obtained from other component models.

To illustrate different representations of operations to obtain the required inputs $I(t_i)$, we first consider a process description with the following code fragment as part of the state transition function f:

head = read("levels").

here, read refers to an operation reading a data set from disc. The values, in this case holding groundwater head values in a catchment, are assigned to the variable head. More generic components can be constructed by specifying the numerical implementations of the physical processes, i.e. the process implementations, independently of spatial and temporal discretisations such as the catchment extent and spatial discretisation or the time step. This makes a component easier to reuse for different catchments or simulation periods. However, using a read operation in the state transition function fixes the assignment of the head values to a read operation from disc. A modeller therefore needs to modify the process implementation in case other sources of the required inputs are available, for instance by another component model that is run simultaneously.

A consistent syntax is required to express input requirements in the process descriptions to allow for a generic process description, independent from an input either obtained as data read from hard disc or calculated by another component model. To provide such syntax, we can apply the function object concept (also referred to as functors) known from programming languages such as C++, Java or Python. Function objects provide a syntax allowing to call an object, which is equivalent to a component model in the field of environmental modelling, with the common syntax of a function call. Function objects can be applied in different contexts without changing the meaning of an operation (see also (Frank, 2005; Mennis, 2010). A modeller can therefore specify an input requirement within the transition function f as function call:

head = groundwater("levels").

where groundwater represents a function object returning a head value. At initialisation of the component model the function object can be parameterised, i.e. the source of the input can be specified according to the current usage of the component model. The modeller is now able to use the component model individually by specifying the function object as disc access, or to apply the component model as part of an integrated model by specifying the function object as a call to another component in an integrated model.

2.2. Execution of data requests

The function object notation introduced above provides modellers with a consistent syntax expressing the origin of data and therefore the data exchange. This is done either as an intrinsic operation of a modelling environment, or by expressing a link to another component in an integrated model. The function syntax provides, in addition to the specification of a data exchange, a means to specify the execution flow of component models (Bulatewicz et al., 2013). A component model (for example landuse) requests a certain input variable from another component model (for example runoff, see Fig. 1 and Table 3 line 18), is paused until the variable has been retrieved, and continues with its process calculation. In multi-scale models with component models having different temporal discretisations, a request-reply based function needs to incorporate time for a correct execution of integrated models. Model components with smaller timesteps use this information to proceed until the common time step with the requesting component.

To illustrate the concept of request-reply based model execution, we consider an integrated model simulating the spatial allocation of bioenergy crops, for instance Eucalyptus. The representation of the system is simplified, as the purpose of the model is mainly to illustrate the concepts of request-reply execution. The integrated model includes three component models: a land use change model, a hydrological model, and a Eucalyptus tree growth model (see Fig. 1). The land use change model spatially allocates expanding Eucalyptus, using a time

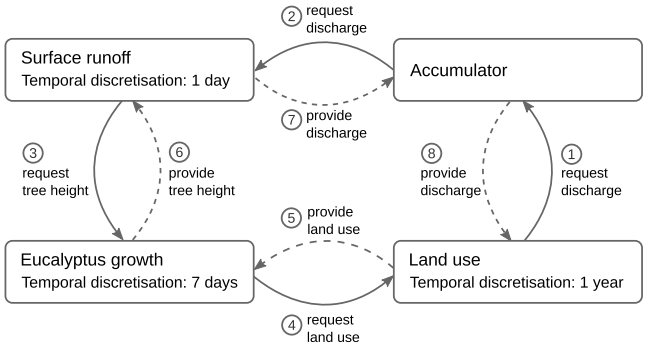

Fig. 1. Conceptual setup of an integrated model simulating the spatial allocation of bioenergy crops. The numbers indicate the order of steps performed in request-reply approach. The land use change model initiates the first input request.

step of a year. Expansion of Eucalyptus will reduce stream discharge, as the water use of Eucalyptus is high. Thus, it is assumed that governmental laws are in place that each year restricts Eucalyptus expansion to subcatchments with a stream discharge of the previous year which was still above a threshold value, i.e. the required environmental flow. To model this, the land use change model requires yearly discharge for all subcatchments for each year. A hydrological model running at a time step of one day is used to calculate the discharge. The land use change model, running at a time step of a year, requests the yearly discharge from an accumulating model component, which converts the daily stream discharge requested from the hydrological model to yearly values. As stream discharge depends on evapotranspiration, which is assumed here to be a function of the tree height, the hydrological model requests the tree height from the Eucalyptus tree growth model, which is the third model component. The tree growth model simulates the growth of the Eucalyptus trees, using a time step of a week. This component requests yearly updates of the map with the simulated Eucalyptus plantations from the land use change model.

A request-reply specification of obtaining input variables directly connects two component models. A subsequent execution of these function calls allows the execution of models with several components, such as the integrated models in the bioenergy example given above. A selected component model thereby adopts the role as main component and initiates the chain of model components. We specify the land use change component model as main component in our example. First, the main component performs its initialisation, and thereby triggers the initialisation of coupled components such as the total runoff component. All subsequent component models will be set to a valid initial state by this recursive initiation. After that, the main component calculates its process descriptions for the first time step, i.e. all operations are repeated until an input variable is required that needs to be provided by a coupled component, here the total runoff component. The function call resembles a query to the total runoff component for a variable corresponding to the current time step t_1 of the land use change component. In general, two alternative situations may arise in requesting data from another component model. In the first situation, the state variable of the external component model corresponding to the time step t_1 has not yet been calculated. The external component therefore needs to execute repeatedly its own time steps until t_1 is reached. For example, the land use change component model with a yearly time step and in need for a runoff value requires the hydrological model to perform 365 time steps to reach a consistent state. Afterwards, the requested variable can be returned. In the second situation, the current time step and thus the state of the external component corresponds to the time step t_1 of the calling component model. The requested variable is already calculated and can be returned to the receiving component immediately.

The subsequent calls and execution of component models are repeated until the main component finishes all time steps.

2.3. Support for uncertainty analysis

Next, we discuss how to deal with uncertainty in integrated models. The objective of uncertainty analysis is to determine probability distributions of stochastic model variables. Monte Carlo simulation is a common computational approach to derive these distributions according to the following procedure (e.g. Heuvelink, 1998):

1. Generate a set of Monte Carlo runs $S = (1, ..., N)$ with realisations for each stochastic parameter and for each stochastic input variable.
2. For each run s in S execute all time steps of the model.
3. Calculate and analyse the ensemble statistics.

This procedure needs to be extended to all components in integrated models, thereby considering that each component model can have its own spatial and temporal discretisation. For a set of $C = (1, ..., M)$ component models in an integrated model, the Monte Carlo simulation is executed as described in Table 1.

To perform a request-based execution of an integrated model, we define a recursive function run_until that calculates the state transition function of a component model c_i until a requested time step t_j. The run_until function initiates a subsequent execution of a coupled component model to obtain an external input, if necessary. Starting from the first timestep of the main component model (e.g. Landuse as c_1 with t_1) and iterating over its timesteps, all M component models will be executed recursively for the whole simulation period.

With this recursive mechanism for a single execution of an integrated model, we can now execute several runs for Monte Carlo simulations. Following the procedure above, the stochastic parameters and the stochastic input variables for all component models are generated first (step 1). All realisations can now be calculated by executing N runs using the run_until function for each time step of the main component model (step 2). Finally, the ensemble statistics are calculated for each component model (step 3).

3. The prototype implementation of the modelling framework

An objective of the modelling framework is to provide model builders with one consistent environment for the construction and analysis of integrated models. We presume that environmental scientists without explicit training in software development are the target audience of our modelling framework. Therefore, we developed a prototype implementation in (Python, 2015), allowing scientists to apply the framework after a short period of familiarisation with the scripting language. The modelling framework provides a set of Python classes that support the construction of coupled spatio-temporal models. The approach presented here is comparable to the one presented by (Karssenberg et al., 2010). Operations available in the PCRaster package (pcrasterEuUrl) can be included to allow for spatial operations and for spatially explicit visualisations of model results.

Table 1
Monte Carlo scheme for the request-reply execution of coupled component models.

```
function run_until(ci, tj):
    if input i from ck required:
        if not ck at tj:
            run_until(ck, tj)
        obtain input i
    while ti < tj:
        calculate state transition function

Step 1:
    for each c in C:
        generate a set s with parameters and inputs suitable for component c

Step 2:
    for each s in S:
        for each time step ti of c1:
            run_until(c1, ti)

Step 3:
    for each c in C
        run postprocessing over all S and time steps of c
```

3.1. Framework classes to build a component model

The modelling framework provides two Python base classes guiding a modeller in the implementation of component models. The DynamicModel base class provides functionality for the development of dynamic component models, and the MonteCarloModel class provides functionality required to perform uncertainty analysis.

3.1.1. The DynamicModel class

The modeller uses Python classes as a basis for a component model, and can add support for dynamic modelling by deriving from the DynamicModel base class. A template script showing the general structure of a component model is given in Table 2. The init section initialises the Model class as a component model with a specified simulation period and time step, and initialises the functionality for executing Monte Carlo simulations, if necessary (lines 3 and 4). The Input class is a supporting construct allowing a modeller to specify the origin of an input variable. This class returns a function object that can be used to express input requests within the process description given in the dynamic section. A modeller can use the Input class either to create a new instance of a coupled component model such as for the WeatherModel (line 5), or to refer to a previously instantiated component model such as the Landuse model (line 6).

The modeller can use the initial method to set the initial value of state variables or parameters (e.g. line 12). With the export method (line 13), a state variable is registered as component output and can therefore be requested by other component models. The contents of the state variables are stored to disc when using the export method.

A modeller implements the operations that are executed for every time step in the dynamic section. Here, data requests to external component models can be specified (lines 16 and 17). The current time step as argument is used by an external component to update its state until it matches the requesting time step, if necessary. The calls return function objects providing access to the latest states of all variables of the queried component model. A modeller can use this object in the remainder of the dynamic section to access the values of specific state variables by using the "dot" notation (i.e. component.stateVariable) such as the precipitation and evapotranspiration values from the weather model (line 18).

Table 2
Python script showing the class template of a component model and the usage of the modelling frameworks.

```
1  class Model(DynamicModel, MonteCarloModel):
2    def __init__(self, start, end, delta, cloneMap):
3      DynamicModel.__init__(self, start, end, delta)
4      MonteCarloModel.__init__(self)
5      self.weather = Input(WeatherModel(start, end, timedelta(days=1)))
6      self.landuse = Input("Landuse")
7
8    def premcloop(self):
9      self.generateRealisations(...)
10
11   def initial(self):
12     self.state = spatial(boolean(0))
13     self.export(self.state, "state")
14
15   def dynamic(self):
16     curr_weather = self.weather(self.current_time_step())
17     curr_lu = self.landuse(self.current_time_step())
18     recharge = curr_weather.precipitation - curr_weather.evapotrans ...
19     self.state = ...
20
21   def postmcloop(self):
22     self.percentiles(...)
23
24 model = Model(datetime(2000, 1, 1), datetime(2010, 1, 1), timedelta(days=1), "clone.map")
25 dynModel = DynamicFramework(model)
26 mcFrw = MonteCarloFramework(dynModel, nrRealisations=50).run()
```

3.1.2. The MonteCarloModel class

The MonteCarloModel base class provides a modeller with the required functionality to execute a model for a given number of realisations. The class requires a modeller to implement the premcloop (Table 2, line 8) and postmcloop (line 21) methods. The premcloop method is executed once for each component model at the start of the simulation. The modeller can use this method, for example, to calculate parameters which are kept constant for all realisations. After all realisations are calculated, the postmcloop method is executed for each component model. Here, modellers can insert operations calculating ensemble statistics from the obtained probability distribution functions, such as variance or quantiles of the stochastic model variables (see also Karssenberg et al., 2010).

Input data and output data are stored for each realisation in separate locations on the hard disc. The MonteCarloModel class provides methods allowing to access this data depending on the currently executed realisation.

3.1.3. Technical access to external component models

The modelling framework allows modellers to express a request-reply mechanism between two interacting component models. To apply this mechanism, the modelling framework needs references to class instances of external component models. These references are required to access component models and to realise the data exchange at model runtime. These references can directly be created by a modeller when instantiating a component model such as is done for the WeatherModel in the component model shown in Table 2 (line 5). The Input class can obtain a reference to the new class instance, and the modelling framework can use this reference to realise component interactions.

Component models, however, do not always obtain direct references to class instances of external component models that need to provide data. This is the case for the bioenergy crop allocation example (Fig. 1). First, the land use component requires data from the hydrological model, and the modeller specifies within the land use component a reference to the total runoff component, and therein a reference to the hydrological component model. The framework can use these given references to access the corresponding component models at the moment of the data request. The tree growth model is the last component model in the initialisation chain and only referenced from the hydrological model. The tree growth model requiring land use as input, however, has no direct reference to the land use component. In this case, the modelling framework cannot directly resolve a reference to the land use model and is thus not able to perform a data request.

To allow a component model to query output variables from any other component model, the DynamicModel class holds a singleton object containing a list with references to all components given in a coupled model. For each initialised component model, an entry is added to the list. A modeller can now use the name, for example “Landuse”, to refer to a component model. The modelling framework can use the given name, the component list and the current time step of the querying component to resolve references and to perform a data request to the external component model.

3.2. Framework classes to execute component models

The modelling framework provides model builders with a class for the construction of component models, and separate classes for the execution of either component models or integrated models. We provide two framework classes for the execution of dynamic models (DynamicFramework) and for Monte Carlo simulations (MonteCarloFramework). The separation of classes for component model implementation and model execution follows the concept presented by (Karssenberg et al., 2010), and allows for a modular development of component models independent of the way of execution. In addition, modellers are able to change between the DynamicFramework and the MonteCarloFramework with limited implementation effort.

The DynamicFramework organises the request-reply execution of a component or coupled models. The framework is instantiated with a DynamicModel, which is the land use change component in our example of the bioenergy crop model. Below, we refer to the component model passed to the DynamicFramework as main component. The run method of the framework starts the request-reply execution by an initial request to the main component model. This initiation of the model run by the DynamicFramework is comparable to the OpenMI v1.4 trigger component (e.g. Becker and Schüttrumpf, 2011; Bulatewicz et al., 2010; Gregersen et al., 2007). First, the initial section of the main component is executed. If necessary, a subsequent initialisation of the coupled components is performed by the main component. After the initialisation of all components in the model, the main component executes its dynamic method, and with it subsequently the dynamic sections of the coupled components.

The MonteCarloFramework execution framework provides the sequential execution of a dynamic model for a given number of independent realisations. The framework takes an instance of the DynamicFramework as first argument, and hence an instance of a component model or an integrated model. The second argument is the number N of realisations to be executed. The framework executes the scheme given in Table 1 with the run method. First, the framework executes the premcloop methods of all component models, and then executes N realisations (50 realisations in Table 2). Finally, the postmcloop methods of all component models are executed.

4. Case study

We now demonstrate the concepts and functionality of the modelling framework prototype by implementing an integrated model. The concepts shown in the previous section are generic and could be used in general with models that can perform initialisation and running of a timestep separately. Here, we use the PCRaster Python module (Karssenberg et al., 2010) to model the spatial processes and build upon the component models and interactions of the bioenergy crop allocation example introduced in Section 2 (Fig. 1). Each of the three domain models, i.e. land use change, vegetation growth and hydrology, is represented by an individual DynamicModel framework class. The land use change, Eucalyptus tree growth and surface runoff components run with yearly, weekly and daily time steps, respectively. The aggregation of the surface runoff is required before it can be used as input to the land use change component, and is thus calculated by a fourth component which does solely aggregation of results, as shown in Fig. 1.

We use a hypothetical 1098 km^2 catchment with an elevation ranging between 100 m and 1896 m and a cell size of 750 × 750 m^2 (see Fig. 2A). The simulation period was set from the years 2010 to 2100. Land use change with random leapfrog development allowing for discontinuous plantings, and surface runoff with random precipitation per daily timestep are represented by stochastic component models. We therefore run a Monte Carlo simulation using 500 realisations to obtain the probability for Eucalyptus tree growing at a particular location in the catchment.

The implementations of the domain specific processes within the components are deliberately kept simplistic to be able to show the entire model scripts. Due to the modular construction of the integrated model, however, component models could be replaced straightforwardly with models holding more realistic process representations. Potential substitutes for the land use change component are, for example, the PLUC model (Verstegen et al., 2012) or the RuimteModel, (Engelen et al., 2011). The PCR-GLOBWB model (e.g. van Beek et al., 2011) could serve as replacement for the hydrological model component.

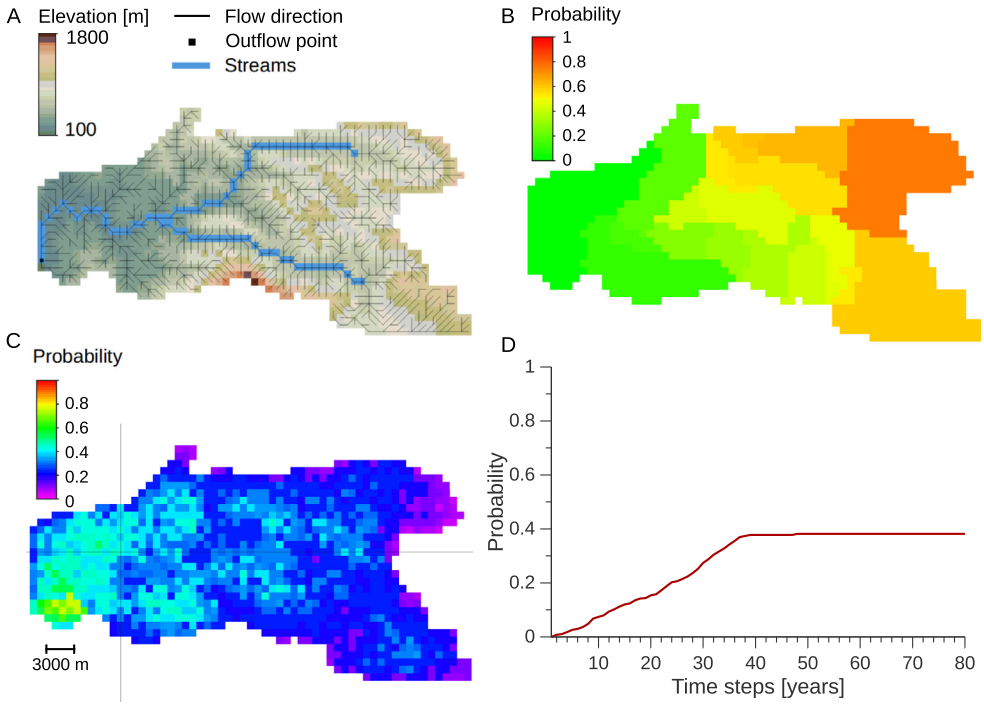

Fig. 2. (A) Topography, river network and flow direction of the catchment. (B) Probability of the cells being excluded from tree allocation (year 2045). (C) Probability of bioenergy crops in 2100. (D) The corresponding time series plot for the selected cell in panel C.

4.1. Domain models

4.1.1. Land use change component

The model script for the land use change component Landuse is shown in Table 3. The land use change component needs yearly-accumulated runoff values to calculate its time step. In the init section, an instance of the TotalRunoff class is created with a daily time step corresponding to the daily time step of the surface runoff component model. The tot_runoff variable therefore allows to obtain the output variable of the TotalRunoff class (line 6).

Variables that are identical for all realisations are initialised in the premcloop section. This method is executed once at start of the Monte Carlo simulation. Here, maps with minimal required discharge values for acceptable bioenergy growing and the main streams in the catchment are read from a file. The local drain direction network, specifying the flow direction for each cell, is derived from the digital elevation model (lines 9–11). The initial section is executed once for each realisation, specifying that no bioenergy plants exist in the catchment as initial condition.

The planting areas for bioenergy crops are allocated once per year by a repeated execution of the dynamic section for each time step. We assume that the allocation of planting areas is based on two conditions. First, new Eucalyptus plantations will occur in the neighbourhood of existing plantations. Second, new Eucalyptus plantations may only be allocated in subcatchments with a stream discharge above a threshold value, i.e. the environmental flow, because Eucalyptus may otherwise further reduce due to its high evapotranspiration the stream discharge to a value below the threshold value. The discharge information is included in the input variable obtained from the TotalRunoff component. The land use component requests the value from the TotalRunoff component and with it initiates a repeated execution of the runoff's dynamic section until the component can provide a variable conforming to the time of the current time step of the land use variable, i.e. for 365 or 366 days, respectively. The obtained yearly runoff is used to determine a map indicating non-suitable growing conditions, excluding the streams in the study area and all cells in a subcatchment having a stream flow below the required threshold. The latter is calculated by the catchment operation, assigning a Boolean false to all cells upstream of stream cells that have a discharge below the threshold (lines 18–20).

Afterwards, the potential growth areas according to the neighbourhood condition are determined. Cells neighbouring existing Eucalyptus plantations are obtained with the window4total operation (line 21). To allow for new planting areas independent of existing plantings, new planting areas can be assigned to the remaining areas with a certain probability. Both suitabilities are combined and assigned to all free cells in the subcatchments, and ordered downwards (lines 22–24). We assume that each year 10 cells with the highest suitability become available for bioenergy crops. Newly assigned areas and existing plantations are then merged to a map holding all cells with Eucalyptus trees (lines 24 and 25).

Table 3
Model script showing the land use change component and the modelling frameworks used. The literal & represents the spatial Boolean AND.

```
1 class Landuse(DynamicModel, MonteCarloModel):
2   def __init__(self, start, end, delta, cloneMap):
3     DynamicModel.__init__(self, start, end, delta)
4     MonteCarloModel.__init__(self)
5     setclone(cloneMap)
6     self.tot_runoff = Input(TotalRunoff(start, end, timedelta(days=1), cloneMap))
7
8   def premcloop(self):
9     self.yearlyRunoffRequired = self.readmap("requiredRunoff")
10    self.mainStreams = self.readmap("streams")
11    self.ldd = lddcreate("dem.map", 1e31, 1e31, 1e31, 1e31)
12
13  def initial(self):
14    self.trees = spatial(boolean(0))
15    self.export(self.trees, "trees")
16
17  def dynamic(self):
18    total_runoff = self.tot_runoff(self.current_time_step())
19    runoffTooLow = self.mainStreams & (total_runoff.totRunoff < self.yearlyRunoffRequired)
20    noNewTrees = catchment(self.ldd, runoffTooLow)
21    suitabilityNeighbourhood = ifthenelse(window4total(scalar(self.trees)) > 0.5, scalar(1), 0)
22    suitabilityRandom = ifthen(pcrnot(self.trees), uniform(1))
23    suitability = ifthen(pcrnot(noNewTrees), (suitabilityNeighbourhood + suitabilityRandom) / 2.0)
24    suitSort = order(0.0 - suitability)
25    self.trees = pcror(self.trees, cover(suitSort < 11, 0))
26    self.export(self.trees, "trees")
27
28  def postmcloop(self):
29    mc_probability("trees", self.sampleNumbers(), self.timeSteps())
30
31 model = Landuse(datetime(2010, 1, 1), datetime(2100, 1, 1), timedelta(days=365), "clone.map")
32 dynModel = DynamicFramework(model)
33 mcFrw = MonteCarloFramework(dynModel, nrRealisations=500).run()
```

The postmcloop section is executed after all realisations were executed. Here, a modeller can insert the operations calculating ensemble statistics, for example to calculate the probability of Eucalyptus planting in each cell with the mc_probability operation (Table 3 line 29). This operation iterates over each time step and each realisation, and returns probability maps for each time step (see Fig. 2).

Lines 31–33 show the instantiation of the land use change component, and consequently all components in the integrated model. Additional arguments of the function are the simulation period specified from 2010 to 2100 with a yearly time step, and a clone map specifying the spatial attributes of the catchment such as extent and number of rows and columns. The model is then executed for 500 Monte Carlo realisations. An individual model run could be achieved by applying the run method to the DynamicFramework instead.

4.1.2. Runoff accumulation

Table 4 shows the implementation to calculate yearly total runoff values. The component is initialised from the land use change component (see Table 3, line 6) with a daily time step and thus matching the temporal discretisation of the hydrological component model. In the init section of the component, a link to the component providing surface runoff is established (line 6). For each daily time step, the variable holding surface runoff is obtained and added (lines 16–17).

The reset method is executed by the modelling framework every time when a coupled component model requires an input variable. This method call is initiated in our integrated model once per year by the land use change component. Then, the variable holding the total runoff values of the previous year is reset to zero (line 9), enabling TotalRunoff to act as an accumulator aggregating over yearly intervals. Afterwards, the TotalRunoff component executes its 365 time steps to proceed until the requested time step of the land use change component.

4.1.3. Hydrological model

The script for the Runoff component modelling the hydrological processes is shown in Table 5. Surface runoff depends on evapotranspiration and therefore the vegetation in the catchment, thus a link to the component providing the tree height is established in the init method (line 6). In the initial section, a map with the local drain directions (Burrough, 1998) is calculated describing the flow network in the catchment. For each time step the vegetation heights are obtained by a request to the tree growth model, and used to calculate the proportion of rainfall reaching the soil surface (lines 14–15), using a linear relation for reasons of brevity. The daily precipitation is assumed as a stochastic input rain ~$N(0.002, 1)$ (line 16), which is then multiplied by the proportion of rainfall netRainProp to determine the generated amount of water reaching the soil surface. This amount is accumulated over the flow network using the accuflux operation to obtain the discharge for each day (lines 17–18).

4.1.4. Tree growth model

Table 6 shows the implementation of the Vegetation component modelling the tree growth of the bioenergy plants. The vegetation growth component is initialised from the runoff component with a weekly time step. We assume a constant growth of the Eucalyptus trees by 2 cm per week (line 13), and a complete removal of biomass in case that the trees exceed a height of 10 m (line 14). Areas that are used for planting of bioenergy crops are calculated in the land use change component, and its latest state is obtained in line 15. This information is used to determine the areas that are not occupied by Eucalyptus trees, where we assume a constant vegetation height of 0.2 m (line 16).

Table 4
Model script for the total runoff accumulation.

```
1  class TotalRunoff(DynamicModel, MonteCarloModel):
2    def __init__(self, start, end, delta, cloneMap):
3      DynamicModel.__init__(self,start, end, delta)
4      MonteCarloModel.__init__(self)
5      setclone(cloneMap)
6      self.runoff = Input(RunoffModel(start, end, ↩
           dt.timedelta(days=1), cloneMap))
7
8    def reset(self):
9      self.totRunoff = scalar(0)
10
11   def initial(self):
12     self.totRunoff = scalar(0)
13     self.export(self.totRunoff, "totRunoff")
14
15   def dynamic(self):
16     runoff_model = self.runoff(self.current_time_step())
17     self.totRunoff = self.totRunoff + runoff_model.runoff
18     self.export(self.totRunoff, "totRunoff")
```

Table 5
Model script to calculate the surface runoff.

```
1  class Runoff(DynamicModel, MonteCarloModel):
2    def __init__(self, start, end, delta, cloneMap):
3      DynamicModel.__init__(self, start, end, delta)
4      MonteCarloModel.__init__(self)
5      setclone(cloneMap)
6      self.vegetation_model = Input(Vegetation(start, end, ↩
           dt.timedelta(days=7), cloneMap))
7
8    def initial(self):
9      self.ldd = lddcreate("dem.map", 1e31, 1e31, 1e31, 1e31)
10     self.runoff = scalar(0)
11     self.export(self.runoff, "runoff")
12
13   def dynamic(self):
14     vegetation = ↩
           self.vegetation_model(self.current_time_step())
15     netRainProp = max(1.0 - (vegetation.vegHeight * 0.1), ↩
           0.0)
16     rain = max(0.002 + normal(1) / 1000.0, 0.0)
17     runoffGen = rain * netRainProp
18     self.runoff = accuflux(self.ldd, runoffGen)
19     self.export(self.runoff, "runoff")
```

Table 6
Model script for the Eucalyptus tree growth component.

```
1  class Vegetation(DynamicModel, MonteCarloModel):
2    def __init__(self, start, end, delta, cloneMap):
3      DynamicModel.__init__(self, start, end, delta)
4      MonteCarloModel.__init__(self)
5      setclone(cloneMap)
6      self.landuse = Input("Landuse")
7
8    def initial(self):
9      self.vegHeight = scalar(0)
10     self.export(self.vegHeight, "vegHeight")
11
12   def dynamic(self):
13     self.vegHeight = self.vegHeight + 0.02
14     self.vegHeight = ifthenelse(self.vegHeight >= 10.0, ↩
           0.0, self.vegHeight)
15     landuse = self.landuse(self.current_time_step())
16     self.vegHeight = ifthenelse(landuse.trees, ↩
           self.vegHeight, 0.2)
17     self.export(self.vegHeight, "vegHeight")
```

4.1.5. Model output results

The model outputs and ensemble statistics can be explored interactively using the Aguila visualisation tool (Pebesma et al., 2007). Fig. 2A shows the topography and flow direction network of the catchment, and the main streams where the discharge is compared with the threshold discharge, i.e. the required environmental flow. The threshold discharge for each cell is assumed to be a fixed proportion of the discharge for the situation without bioenergy. Note that the streams shown in blue increase in size from right to left, due to an increase in catchment area from right to left. Fig. 2B, Fig. 2C and Fig. 2D show exemplary ensemble results calculated from 500 realisations of the Monte Carlo simulation. Fig. 2B shows, for the year 2045, the probability of a cell being excluded from bioenergy allocation due to a stream discharge below the threshold value in one of its downstream cells. The map shows that cells flowing into smaller streams (right side of the map) are more likely to be excluded than cells that flow into larger streams (left side of the map). This is due to spatial differences in the relative impact of Eucalyptus on stream discharge. Allocating new Eucalyptus to a cell will reduce the stream discharge in all stream cells downstream of the cell (not shown in Fig. 2). In the downstream part of the catchment, this has a small relative effect on the discharge in the main streams (shown in blue in Fig. 2A), because streams here are large. As a result, for the majority of the realisations the stream discharge in the downstream reaches of the streams will not fall below the threshold value. This results in a low probability for excluding cells from allocation of bioenergy in the downstream region of the catchment. Fig. 2C gives the map with probabilities of bioenergy in the catchment for the year 2100, which are for the same reason highest in the downstream area. The probability maps are calculated for each year with the mc_probability method (see Table 3). Fig. 2D shows the probability of Eucalyptus allocation in the selected cell as a time series plot generated by the interactive visualisation of Aguila.

5. Discussion and conclusion

We presented the general conditions for a request-reply execution and analysis of component-based integrated models. A syntax of function objects is proposed as a uniform notation for the modeller to express data requests from external input sources, whether the source is a data file or output from another component model. The developed modelling framework provides a set of class templates supporting the implementation of process descriptions supporting a function-based execution of component models. We demonstrated the usage of the modelling framework by means of an integrated model for bioenergy crop allocation and presented results from an uncertainty analysis.

In general, different operations are needed to express data import within the transition function, such as an operation to read data from hard disc, or to specify a link to a coupled component model. By using the function object notation to call objects providing the requested input, the manner of obtaining the data is transferred from the process implementation to the input interface of a model component. A common notation to express data requests enables a modeller to switch more easily between different input sources without modifying the component's process. The process implementation depends less on the type of an input data source, therefore making the component model more independent and consequently the component more reusable in different modelling contexts. A concise notation for data requirements also resembles the imperative way to specify the state transition function f (Eq. (1)): couplings to other components can be expressed in the same way as using intrinsic operations of the modelling environment. Using data request functions allows the modeller to specify explicitly the execution flow of a coupled model. The chronological execution of component models in an integrated model may therefore be more transparent than in an execution flow automatically generated by a scheduling system (e.g. Schmitz et al., 2013).

The modelling framework provides with the reset section functionality to re-initialise state variables at model runtime. This enables modellers to convert the role of a component model to an accumulator. By specifying aggregating processes, modellers can implement building blocks bridging temporal discretisation differences. We demonstrated the temporal bridging in the case study with the TotalRunoff class. Spatial discretisation differences between component models, however, are the second important reason for adapting variables prior to exchange. The modelling framework supports this application case and can be used to change the spatial attributes of component models while executing their process implementations. A modeller can implement the dynamic section in such a way that for each time step first the setclone function is called to match the spatial discretisation of the incoming variables. Then, variables can be resampled to a different spatial discretisation required by another component model, for instance by using PCRaster or GDAL (GDAL Development Team, 2016) functionality. Afterwards, by using the setclone again, the spatial discretisation of the component matches the one of the outgoing variables.

We demonstrated the construction of an integrated model by purely using the Python language. To integrate components developed in other programming languages, several wrapping techniques are feasible. By using language bindings, existing Fortran or C++ code can be made accessible as Python operations and modules. An example for this approach is the accuflux operation of PCRaster used in Table 5. Another approach is the development of additional wrapper code that provides interoperability to other (domain specific) languages, as demonstrated for example by Bulatewicz et al. (2013). Alternatively, external applications can be integrated via file access and system calls (e.g. Hahn et al., 2009). However, wrapping external models requires additional implementation efforts from the model builder as input and output files need to be converted to Python data types such that these can be processed by the modelling framework.

Uncertainty analysis allows environmental modellers to assess the errors in model outputs resulting from uncertainty in model inputs, model parameters or the model structure itself (e.g. Heuvelink, 1998; Karssenberg and de Jong, 2005). The quantification of uncertainty is recommended as part of the good modelling practises (e.g. Jakeman et al., 2006; Refsgaard et al., 2007). However, assessing the uncertainty of integrated models is more complex, as potentially several domain specific processes and therefore component models are included, and these can operate on their individual spatial and temporal discretisations. By providing the premcloop and postmcloop sections in our framework, probability distributions can be straightforwardly obtained for each component model as demonstrated for the Landuse component model (Fig. 2). By following a modular model development approach, accumulating components such as the TotalRunoff component could be replaced to test different aggregation methods. A software framework including support for Monte Carlo simulation makes it in our opinion easier for domain specialists to evaluate feedback effects, aggregation methods and the influence of multiple spatial and temporal scales in integrated models (see also Elag et al., 2011; Ewert et al., 2009).

The framework prototype provides one environment for the construction and assessment of spatio-temporal multi-scale models with an implementation language that environmental scientists can use after a limited time of familiarisation. Still, several issues require further attention in the development of the modelling framework.

One concern is the efficiency of the current implementation of the modelling framework. At this stage, the complete history of state variables is stored to the hard disc, allowing to fulfil arbitrary data requests from any component in the integrated model. Storing the state variables of all component models for each time step and realisation, however, may require significant amounts of disc space as large spatial data sets are involved. Maintaining the complete history of the state variables requires additional memory for bookkeeping and additional runtime for querying the history, although the organisational overhead is limited. Furthermore, the implementation calculating the yearly runoff (see

Table 4) also stores the aggregated variable for each time step. As these intermediate values are of no particular use for other component models, storing only the yearly aggregated variable could reduce storage consumption. Potential of further storage reduction could be explored by in-memory storage of the state variable history, comparable to OpenMI's SmartBuffer (see, e.g. Elag et al., 2011). Algorithmic improvements can be implemented discarding the total history in case of equal time steps when only the previous state variable is required.

The modelling framework supports a concurrent execution of Monte Carlo realisations to exploit all CPU resources on multi-core computers. Still, instances of component models and the model execution of a realisation are maintained in one computational thread. The component models are executed sequentially, although their implementations are in general independent of each other and could be executed concurrently. For large spatio-temporal models, a single-threaded execution can result in long model runtimes. In addition, a single-threaded execution limits either the number of components in a model, or the size of the spatial data used by the components due to memory limitations constrained by the operating system and the hardware used.

The usage of one single component model class and the autonomy of the component model classes from a specific way of execution allow a modeller to run single component and coupled models in dynamic or Monte Carlo simulations. A modeller can straightforwardly select and assess alternative model compositions, and is therefore assisted with the exploratory, plug-and-play modelling using reusable component models (e.g. de Kok et al., 2015; Papajorgji, 2005). The modular architecture of the modelling environment allows adding additional optimisation techniques such as data assimilation (e.g. Evensen, 2003; Karssenberg et al., 2010; Moradkhani et al., 2005). Technically, the reset method could also be used by the modelling framework to perform the required state update, when data assimilation approaches such as the particle filter is used (e.g. Doucet et al., 2001; van Leeuwen, 2003). Still, observational data may be available for various component models and different time steps. Defining appropriate filter moments in multi-scale component models to understand and to quantify the influence of observational data remains a challenge for an environmental modeller.

Acknowledgements

This work was supported by research funding from the Flemish Institute for Technological Research (VITO, Belgium). We thank Kor de Jong (Utrecht University) and Guy Engelen (VITO) for their inputs to the work presented in this paper. Finally, we thank the anonymous reviewers for their useful comments to the manuscript.

References

ArcGIS, 2015. Environmental Systems Research Institute. http://www.esri.com/ Accessed 5 October 2015.

Argent, R.M., 2005. A case study of environmental modelling and simulation using transplantable components. Environ. Model. Softw. 20, 1514–1523. http://dx.doi.org/10.1016/j.envsoft.2004.08.016.

Construction and Evaluation of Models of Environmental Systems. In: Beck, M.B., Jakeman, A.J., McAleer, M.J., Beck, M.B., Jakeman, A.J., McAleer, M.J. (Eds.), Modelling Change in Environmental Systems. John Wiley & Sons Ltd., New York, pp. 3–35.

Becker, B.P., Schüttrumpf, H., 2011. An OpenMI module for the groundwater flow simulation programme Feflow. J. Hydroinf. 13, 1–12. http://dx.doi.org/10.102166/hydro.2010.039.

van Beek, L.P.H., Wada, Y., Bierkens, M.F.P., 2011. Global monthly water stress: 1. Water balance and water availability. Water Resour. Res. 47, W07517. http://dx.doi.org/10.1029/2010WR009791.

Beven, K., Binley, A., 1992. The future of distributed models: Model calibration and uncertainty prediction. Hydrol. Process. 6, 279–298. http://dx.doi.org/10.1002/hyp.3360060305.

Booch, G., Maksimchuk, R.A., Engel, M.W., Young, B.J., Conallen, J., Houston, K.A., 2007. Object Oriented Analysis and Design with Applications Addison Wesley.

Bulatewicz, T., Allen, A., Peterson, J.M., Staggenborg, S., Welch, S.M., Steward, D.R., 2013. The Simple Script Wrapper for OpenMI: Enabling interdisciplinary modeling studies. Environ. Model. Softw. 39, 283–294. http://dx.doi.org/10.1016/j.envsoft.2012.07.006.

Bulatewicz, T., Yang, X., Peterson, J.M., Staggenborg, S., Welch, S.M., Steward, D.R., 2010. Accessible integration of agriculture, groundwater, and economic models using the Open Modeling Interface (OpenMI): Methodology and initial results. Hydrol. Earth Syst. Sci. 14, 521–534. http://dx.doi.org/10.5194/hess-14-521-2010.

Burrough, P.A., 1998. Dynamic Modelling and Geocomputation. In: Longley, P.A., Brooks, S.M., McDonnel, R., MacMillan, B. (Eds.), Geocomputation: A Primer. Wiley, Chichester, pp. 165–191.

Claessens, L., Schoorl, J.M., Verburg, P.H., Geraedts, L., Veldkamp, A., 2009. Modelling interactions and feedback mechanisms between land use change and landscape processes. Agric. Ecosyst. Environ. 129, 157–170. http://dx.doi.org/10.1016/j.agee.2008.08.008.

David, O., Ascough II, J.C., Lloyd, W., Green, T.R., Rojas, K.W., Leavesley, G.H., Ahuja, L.R., 2013. A software engineering perspective on environmental modeling framework design: The object modeling system. Environ. Model. Softw. 39, 201–213. http://dx.doi.org/10.1016/j.envsoft.2012.03.006.

Del Giudice, D., Reichert, P., Bareš, V., Albert, C., Rieckermann, J., 2015. Model bias and complexity – Understanding the effects of structural deficits and input errors on runoff predictions. Environ. Model. Softw. 64, 205–214. http://dx.doi.org/10.1016/j.envsoft.2014.11.006.

Doucet, A., de Freitas, N., Gordon, N., 2001. Sequential Monte Carlo Methods in Practice. Statistics for Engineering and Information Science. Springer, New York.

Elag, M.M., Goodall, J.L., Castronova, A.M., 2011. Feedback loops and temporal misalignment in component-based hydrologic modeling. Water Resour. Res. 47, W12520. http://dx.doi.org/10.1029/2011WR010792.

Engelen, G., Poelmans, L., Uljee, I., De Kok, J.L., Van Esch, L., 2011. De Vlaamse Ruimte in 4 Wereldbeelden - Scenarioverkenning 2050 - Kwantitatieve Verwerking (The Flemish Space in 4 World Views – Scenario Exploration 2050 - Quantification). A Study for the Policy Research Center for Spatial Planning and Housing. VITO Report 2011/RMA/R/363. Flemish Institute for Technological Research (VITO), Mol, Belgium (In Dutch).

Evensen, G., 2003. The ensemble Kalman filter: Theoretical formulation and practical implementation. Ocean Dyn. 53, 343–367. http://dx.doi.org/10.1007/s10236-003-0036-9.

Ewert, F., van Ittersum, M.K., Bezlepkina, I., Therond, O., Andersen, E., Belhouchette, H., Bockstaller, C., Brouwer, F., Heckelei, T., Janssen, S., Knapen, R., Kuiper, M., Louhichi, K., Olsson, J.A., Turpin, N., Wery, J., Wien, J.E., Wolf, J., 2009. A methodology for enhanced flexibility of integrated assessment in agriculture. Environ. Sci. Pol. 12, 546–561. http://dx.doi.org/10.1016/j.envsci.2009.02.005.

ExtendSim, 2015. Imagine that Product Website. http://www.extendsim.com/ accessed 5 October 2015.

Frank, A.U., 2005. Map Algebra Extended with Functors for Temporal Data. Perspectives in Conceptual Modelingvolume 3770 of Lect. Notes Comput. Sci.Springer, Berlin Heidelberg, pp. 194–207 http://dx.doi.org/10.1007/11568346_22.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software Addison Wesley.

GDAL Development Team, 2016. GDAL – Geospatial Data Abstraction Library. http://www.gdal.org/ Accessed 13 January 2016.

Gregersen, J.B., Gijsbers, P.J.A., Westen, S.J.P., 2007. OpenMI: Open modelling interface. J. Hydroinf. 9, 175–191. http://dx.doi.org/10.2166/hydro.2007.023.

Hahn, J.B., Kofalk, S., De Kok, J.L., Berlekamp, J., Evers, M., 2009. Elbe DSS: A Planning Support System for Strategic River Basin Planning. In: Geertman, S., Stillwell, J. (Eds.), Planning Support Systems Best Practice and New Methods. Springer, Netherlands, pp. 113–136 http://dx.doi.org/10.1007/978-1-4020-8952-7_6 (chapter 6).

Hamilton, S.H., ElSawah, S., Guillaume, J.H.A., Jakeman, A.J., Pierce, S.A., 2015. Integrated assessment and modelling: Overview and synthesis of salient dimensions. Environ. Model. Softw. 64, 215–229. http://dx.doi.org/10.1016/j.envsoft.2014.12.005.

Heuvelink, G.B.M., 1998. Error Propagation in Environmental Modelling with GIS. Taylor & Francis, London.

Hill, C., DeLuca, C., Balaji, V., Suarez, M., Da Silva, A., 2004. The Architecture of the Earth System Modeling Framework. Comput. Sci. Eng. 6, 18–28. http://dx.doi.org/10.1109/MCISE.2004.1255817.

Hinkel, J., 2009. The PIAM approach to modular integrated assessment modelling. Environ. Model. Softw. 24, 739–748. http://dx.doi.org/10.1016/j.envsoft.2008.11.005.

Holzworth, D.P., Huth, N.I., de Voil, P.G., 2010. Simplifying environmental model reuse. Environ. Model. Softw. 25, 269–275. http://dx.doi.org/10.1016/j.envsoft.2008.10.018.

Huang, J., Gao, J., Xu, Y., Liu, J., 2015. Towards better environmental software for spatio-temporal ecological models: Lessons from developing an intelligent system supporting phytoplankton prediction in lakes. Ecol. Inf. 25, 49–56. http://dx.doi.org/10.1016/j.ecoinf.2014.11.005.

van Ittersum, M.K., Ewert, F., Heckelei, T., Wery, J., Alkan Olsson, J., Andersen, E., Bezlepkina, I., Brouwer, F., Donatelli, M., Flichman, G., Olsson, L., Rizzoli, A.E., van der Wal, T., Wien, J.E., Wolf, J., 2008. Integrated assessment of agricultural systems – A component-based framework for the European Union (SEAMLESS). Agric. Syst. 96, 150–165. http://dx.doi.org/10.1016/j.agsy.2007.07.009.

Jakeman, A.J., Letcher, R.A., Norton, J.P., 2006. Ten iterative steps in development and evaluation of environmental models. Environ. Model. Softw. 21, 602–614. http://dx.doi.org/10.1016/j.envsoft.2006.01.004.

Jørgensen, S.E., Bendoricchio, G., 2001. Fundamentals of Ecological Modelling. Elsevier, Amsterdam.

Karssenberg, D., 2002. The value of environmental modelling languages for building distributed hydrological models. Hydrol. Process. 16, 2751–2766. http://dx.doi.org/10.1002/hyp.1068.

Karssenberg, D., de Jong, K., 2005. Dynamic environmental modelling in GIS: 2. Modelling error propagation. Int. J. Geogr. Inf. Sci. 19, 623–637. http://dx.doi.org/10.1080/13658810500104799.

Karssenberg, D., de Jong, K., van der Kwast, J., 2007. Modelling landscape dynamics with Python. Int. J. Geogr. Inf. Sci. 21, 483–495. http://dx.doi.org/10.1080/13658810601063936.

Karssenberg, D., Schmitz, O., Salamon, P., de Jong, K., Bierkens, M.F.P., 2010. A software framework for construction of process-based stochastic spatio-temporal models and data assimilation. Environ. Model. Softw. 25, 489–502. http://dx.doi.org/10.1016/j.envsoft.2009.10.004.

Kelly (Letcher), R.A., Jakeman, A.J., Barreteau, O., Borsuk, M.E., ElSawah, S., Hamilton, S.H., Henriksen, H.J., Kuikka, S., Maier, H.R., Rizzoli, A.E., van Delden, H., Voinov, A.A., 2013. Selecting among five common modelling approaches for integrated environmental assessment and management. Environ. Model. Softw. 47, 159–181. http://dx.doi.org/10.1016/j.envsoft.2013.05.005.

de Kok, J.L., Engelen, G., Maes, J., 2015. Reusability of model components for environmental simulation – Case studies for integrated coastal zone management. Environ. Model. Softw. 68, 42–54. http://dx.doi.org/10.1016/j.envsoft.2015.02.001.

van Leeuwen, P.J., 2003. A variance-minimizing filter for large-scale applications. Mon. Weather Rev. 131, 2071–2084. http://dx.doi.org/10.1175/1520-0493(2003)131<2071:AVFFLA>2.0.CO;2.

Lindenschmidt, K.E., Fleischbein, K., Baborowski, M., 2007. Structural uncertainty in a river water quality modelling system. Ecol. Model. 204, 289–300. http://dx.doi.org/10.1016/j.ecolmodel.2007.01.004.

Liu, J.S., Chen, R., 1998. Sequential Monte Carlo methods for dynamic systems. J. Am. Stat. Assoc. 93, 1032–1044. http://dx.doi.org/10.2307/2669847.

Marohn, C., Schreinemachers, P., Quang, D.V., Berger, T., Siripalangkanont, P., Nguyen, T.T., Cadisch, G., 2013. A software coupling approach to assess low-cost soil conservation strategies for highland agriculture in Vietnam. Environ. Model. Softw. 45, 116–128. http://dx.doi.org/10.1016/j.envsoft.2012.03.020.

Matott, L.S., Babendreier, J.E., Purucker, S.T., 2009. Evaluating uncertainty in integrated environmental models: A review of concepts and tools. Water Resour. Res. 45, W06421. http://dx.doi.org/10.1029/2008WR007301.

Mennis, J., 2010. Multidimensional map algebra: design and implementation of a spatio-temporal GIS processing language. Trans. GIS 14, 1–21. http://dx.doi.org/10.1111/j.1467-9671.2009.01179.x.

Moore, R.V., Tindall, C.I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Pol. 8, 279–286. http://dx.doi.org/10.1016/j.envsci.2005.03.009.

Moradkhani, H., Hsu, K.L., Gupta, H., Sorooshian, S., 2005. Uncertainty assessment of hydrologic model states and parameters: Sequential data assimilation using the particle filter. Water Resour. Res. 41, W05012. http://dx.doi.org/10.1029/2004WR003604.

Muth Jr., D.J., Bryden, K.M., 2013. An integrated model for assessment of sustainable agricultural residue removal limits for bioenergy systems. Environ. Model. Softw. 39, 50–69. http://dx.doi.org/10.1016/j.envsoft.2012.04.006.

Neteler, M., Bowman, M.H., Landa, M., Metz, M., 2012. GRASS GIS: A multi-purpose open source GIS. Environ. Model. Softw. 31, 124–130. http://dx.doi.org/10.1016/j.envsoft.2011.11.014.

Papajorgji, P., 2005. A plug and play approach for developing environmental models. Environ. Model. Softw. 20, 1353–1357. http://dx.doi.org/10.1016/j.envsoft.2004.11.016.

PCRaster, 2015. PCRaster website. http://www.pcraster.eu (Accessed 5 October 2015).

Pebesma, E.J., de Jong, K., Briggs, D., 2007. Interactive visualization of uncertain spatial and spatio-temporal data under different scenarios: An air quality example. Int. J. Geogr. Inf. Sci. 21, 515–527. http://dx.doi.org/10.1080/13658810601064009.

Peckham, S.D., Hutton, E.W.H., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: The design of CSDMS. Comput. Geosci. 53, 3–12. http://dx.doi.org/10.1016/j.cageo.2012.04.002.

Pullar, D., 2004. SimuMap: A computational system for spatial modelling. Environ. Model. Softw. 19, 235–243. http://dx.doi.org/10.1016/S1364-8152(03)00151-8.

Python, 2015. Python Programming Language Website. http://www.python.org/ (Accessed 5 October 2015).

QGIS Development Team, 2015. QGIS Geographic Information System. Open Source Geospatial Foundationhttp://qgis.osgeo.org (Accessed 5 October 2015).

Refsgaard, J.C., van der Sluijs, J.P., Brown, J., van der Keur, P., 2006. A framework for dealing with uncertainty due to model structure error. Adv. Water Resour. 29, 1586–1597. http://dx.doi.org/10.1016/j.advwatres.2005.11.013.

Refsgaard, J.C., van der Sluijs, J.P., Højberg, A.L., Vanrolleghem, P.A., 2007. Uncertainty in the environmental modelling process – A framework and guidance. Environ. Model. Softw. 22, 1543–1556. http://dx.doi.org/10.1016/j.envsoft.2007.02.004.

Risbey, J., Sluijs, J., Kloprogge, P., Ravetz, J., Funtowicz, S., Corral Quintana, S., 2005. Application of a checklist for quality assistance in environmental modelling to an energy model. Environ. Model. Assess. 10, 63–79. http://dx.doi.org/10.1007/s10666-004-4267-z.

Rizzoli, A.E., Donatelli, M., Athanasiadis, I.N., Villa, F., Huber, D., 2008. Semantic links in integrated modelling frameworks. Math. Comput. Simul. 78, 412–423. http://dx.doi.org/10.1016/j.matcom.2008.01.017.

Roberts, J.J., Best, B.D., Dunn, D.C., Treml, E.A., Halpin, P.N., 2010. Marine geospatial ecology tools: An integrated framework for ecological geoprocessing with ArcGIS, python, R, MATLAB, and C++. Environ. Model. Softw. 25, 1197–1207. http://dx.doi.org/10.1016/j.envsoft.2010.03.029.

Rotmans, J., 1990. IMAGE: An Integrated Model to Assess the Greenhouse Effect. Kluwer.

Schmitz, O., Karssenberg, D., de Jong, K., de Kok, J.L., de Jong, S.M., 2013. Map algebra and model algebra for integrated model building. Environ. Model. Softw. 48, 113–128. http://dx.doi.org/10.1016/j.envsoft.2013.06.009.

Schwanghart, W., Kuhn, N.J., 2010. TopoToolbox: A set of matlab functions for topographic analysis. Environ. Model. Softw. 25, 770–781. http://dx.doi.org/10.1016/j.envsoft.2009.12.002.

Schweitzer, C., Priess, J.A., Das, S., 2011. A generic framework for land-use modelling. Environ. Model. Softw. 26, 1052–1055. http://dx.doi.org/10.1016/j.envsoft.2011.02.016.

Shen, C., Niu, J., Fang, K., 2014. Quantifying the effects of data integration algorithms on the outcomes of a subsurface–land surface processes model. Environ. Model. Softw. 59, 146–161. http://dx.doi.org/10.1016/j.envsoft.2014.05.006.

Sklar, E., 2007. Software review: NetLogo, a multi-agent simulation environment. Artif. Life 13, 303–311. http://dx.doi.org/10.1162/artl.2007.13.3.303.

Steiniger, S., Bocher, E., 2009. An overview on current free and open source desktop GIS developments. Int. J. Geogr. Inf. Sci. 23, 1345–1370. http://dx.doi.org/10.1080/13658810802634956.

Szyperski, C., 2002. Component Software. Beyond Object-Oriented Programming, second ed. Addison Wesley.

Tomlin, C.D., 1990. Geographic Information Systems and Cartographic Modeling. Prentice Hall, Englewood Cliffs, NJ.

Verburg, P., 2006. Simulating feedbacks in land use and land cover change models. Landsc. Ecol. 21, 1171–1183. http://dx.doi.org/10.1007/s10980-006-0029-4.

Verstegen, J.A., Karssenberg, D., van der Hilst, F., Faaij, A., 2012. Spatio-temporal uncertainty in Spatial Decision Support Systems: A case study of changing land availability for bioenergy crops in Mozambique. Comput. Environ. Urban. Syst. 36, 30–46. http://dx.doi.org/10.1016/j.compenvurbsys.2011.08.003.

Villa, F., Costanza, R., 2000. Design of multi-paradigm integrating modelling tools for ecological research. Environ. Model. Softw. 15, 169–177. http://dx.doi.org/10.1016/S1364-8152(99)00032-8.

Welsh, W.D., Vaze, J., Dutta, D., Rassam, D., Rahman, J.M., Jolly, I.D., Wallbrink, P., Podger, G.M., Bethune, M., Hardy, M.J., Teng, J., Lerat, J., 2013. An integrated modelling framework for regulated river systems. Environ. Model. Softw. 39, 81–102. http://dx.doi.org/10.1016/j.envsoft.2012.02.022.