Environmental Modelling & Software 48 (2013) 113–128

Contents lists available at SciVerse ScienceDirect

Environmental Modelling & Software

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

Map algebra and model algebra for integrated model building

Oliver Schmitz [a,b,*], Derek Karssenberg [a], Kor de Jong [a], Jean-Luc de Kok [b], Steven M. de Jong [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 21 September 2012
Received in revised form
23 May 2013
Accepted 21 June 2013
Available online

Keywords:
Component-based modelling
Spatio-temporal simulation
Biomass-harvest model
Python
PCRaster

ABSTRACT

Computer models are important tools for the assessment of environmental systems. A seamless workflow of construction and coupling of model components is essential for environmental scientists. However, currently available software packages are often tailored either to the construction of model components, or to the coupling of existing components. Combining both objectives is not straightforward, because it requires merging concepts for model component building and model component coupling. Also, software packages should be usable for domain experts such as hydrologists or ecologists who do not necessarily have expert knowledge in programming.

We propose an integrated modelling framework that provides descriptive means to specify (1) model components with conventional map algebra, and (2) interactions between model components with model algebra. A prototype implementation in a high-level scripting language supports the building of integrated spatio-temporal models. For a seamless coupling of model components with different temporal and spatial discretisation, we introduce the use of accumulators. These handle the temporal aggregation of model component outputs. The framework provides templates for the custom construction of model components and accumulators, and a management layer arranges the schedule for the execution of the integrated model. We use the prototype implementation of the framework in an illustrative case study to build an integrated model that couples model components simulating the interaction between biomass growth, wildfire, and human impacts with different temporal discretisations. The high-level Python language is used as model building environment to allow domain experts without in-depth knowledge of software development practices to conduct exploratory model construction and analysis.

Software availability

Details about the modelling framework prototype and the case study data set can be found at the PCRaster website
Contact: o.schmitz@uu.nl
Required software:
PCRaster: Linux, OS X and Windows, GNU GPL v3, available at http://www.pcraster.eu/
Python: all major platforms, available at http://www.python.org/

1. Introduction

Evaluating the consequences of environmental phenomena such as the impacts of climate change or the scarcity of resources requires accounting of multiple processes, interactions and feedbacks. Computer models can help to develop a better understanding of these complex systems. However, building these models relies on scientific knowledge from a multitude of domains, and requires integration of this knowledge. The usage of a modular structure is thereby desired to avoid integrating the expertise of scientists from these domains into one large monolithic model. Hence, segmenting a complex system into manageable parts helps to reduce the overall complexity of both the modelling process and the model. The assembly of multiple model components from different domains into a coupled system is known as integrated modelling (Argent, 2004; Hinkel, 2009; Laniak et al., 2013).

Formulating models that simulate changes in spatial environments is an evolutionary process, which is influenced by various

* Corresponding author. 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).

1364-8152/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.envsoft.2013.06.009

factors. For example, scientific advance or the testing of a new hypothesis are factors that can affect the process descriptions in model components or the composition and assembly of integrated models. Also, technical advances such as new remote sensing products delivering data sets in higher resolution may allow the refinement of spatial processes in the model. To benefit from these advances, a flexible modelling environment is required that allows seamless modification of model components and their integration. The modeller needs to be equipped with a reliable and comprehensible way to quickly build new model components, to maintain and extend existing model components, and to couple those depending on the modelling objective.

To assist model builders in the development of integrated models a variety of software applications are available. For instance, domain specific packages provide integrated functionality such as intrinsic data types and operations thereon which can be used to construct model components. Examples are MapScript (Pullar, 2003) or PCRaster (Wesseling et al., 1996) for field-based modelling, NetLogo (Sklar, 2007) or Repast (North et al., 2006) for agent-based modelling, or the General Algebraic Modeling System (Brooke et al., 1998) for the development of economic models. Other software packages are directed towards the coupling of existing model components providing standardised interfaces to describe and transfer data. Examples are the Typed Data Transfer library (Hinkel, 2009) or the Open Modelling Interface (OpenMI, Gregersen et al., 2007). These packages require the modeller to become skilled in at least two, but possibly more, different software environments. Extending existing system programming languages such as Fortran, C++ or C# with functionality dedicated to the development of integrated models such as in the ESMF (e.g. Collins et al., 2005), Geonamica (e.g. van Delden et al., 2007), ENVISION (e.g. Bolte et al., 2007) or E2 (Argent et al., 2009) facilitates both the construction and coupling of model components. These software tools support modellers in the construction of integrated models by building modular components, and subsequent coupling of these components. Resolving the discrepancies resulting from the coupling of components with different spatial or temporal discretisations, however, remains a problem for a modeller.

We propose a new modelling environment to describe geospatial model components and to couple these components to integrated systems, providing a means to bridge differences in spatio-temporal discretisation between components. We combine the two concepts of map algebra and model algebra. Spatio-temporal map algebra as conventional concept (e.g. Tomlin, 1990; Wesseling et al., 1996) provides a means for model builders to program model components that represent processes acting in a subdomain of the environmental system. We introduce a model algebra to build large systems by coupling these model components. Using the model algebra, the model builder defines component interactions allowing for a coupling of model components with different spatio-temporal discretisations. We propose an accumulator building block used to align model components with different spatial and temporal characteristics. We show a software prototype implemented in Python using the PCRaster module (Karssenberg et al., 2007). The software prototype provides templates for model components and accumulators as building blocks for coupled models.

First, we introduce the model component as a building block for integrated models, and then introduce map algebra as a generic instrument to describe spatial processes incorporated in the model component. In Section 3, we illustrate the coupling of model components with different discretisations by means of accumulators. We introduce Python as a modelling environment in Section 4 and discuss the technical design enabling the ordered execution of components and accumulators. Section 5 describes a case study with simplified models describing biomass growth and effects of wildfire and human impacts, which is used to demonstrate the building of model components, their coupling to integrated models, and the modification and extension of these integrated models. A discussion on the benefits of this unified modelling framework for integrated modelling concludes this paper.

2. Building integrated models

First, we outline the general concepts of a model building block and the spatio-temporal map algebra used to express spatio-temporal processes on fields. Next, model algebra is introduced for the coupling of those building blocks with respect to different temporal discretisations.

2.1. Map algebra

A model component is a building block for the construction of integrated models and holds the numerical descriptions that represent a particular environmental process. Fig. 1 shows the conceptual separation of a model component into internal and external functionality. This separation avoids a monolithic setup of integrated models and assists in a modular development of reusable components (c.f. Voinov et al., 2004; Argent, 2005; Papajorgji, 2005; Rizzoli et al., 2008).

The external interface of a model component consists of the definition of the input and output variables. $I(t_i)$ is a set of input variables at time step t_i required for the model component to proceed its calculation. $O(t_i)$ refers to the set of output variables. These are accessible for other components and accumulators.

The process implementation of the model component is encapsulated and not visible for other model components. Starting from a set of initial states Z_0 and a set of parameter values P, a transition function f transfers the state variables Z from time step t_{i-1} into new state variables in the subsequent time step t_i according to the following equation (Beck et al., 1993; Burrough, 1998):

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

The dynamic behaviour of the model component is simulated by iterating the state transition function f over discrete time steps. The model builder can formulate the transition function to simulate non-spatial phenomena such as lumped models, or to simulate spatially distributed processes. The complexity of f can be rather limited as for example in the calculation of a topographical slope. In most cases, however, model components will include descriptions that are more complex, such as groundwater processes requiring several input variables and providing multiple output variables.

To describe the environmental processes, a domain specialist who is considered to be the model builder needs to be provided with a means to program the transition function of Equation (1). A

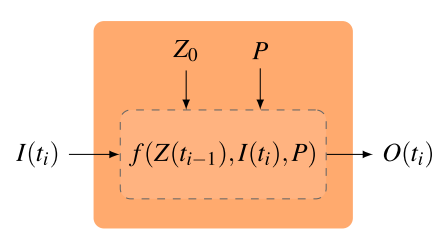

Fig. 1. Conceptualisation of a model component. The external interface describes the required input variables I and provided output variables O per time step t_i. The implementation of the modelled process is internal. Starting from an initial state Z_0, the transition function f transfers the component state Z into a new state. P is a set of parameters.

conventional approach is based on the map algebra and cartographic modelling introduced by Tomlin (1990). Map algebra provides data representation and processing constructs that are easily understood by typical end users (Pullar, 2003) enabling the model builder to express process descriptions in a formal way similar to a mathematical notation. Map algebra therefore allows the description of model components in an objective-focused rather than implementation-focused manner.

The map algebra operations can be classified as follows (c.f. Tomlin, 1990; Karssenberg and de Jong, 2005; Schmitz et al., 2009) (Fig. 2):

(A) Point or local operations affecting the attributes of individual cells. For example, logical and arithmetic operators belong to this category.
(B) Direct neighbourhood operations regarding a restricted spatial neighbourhood such as von Neumann or Moore neighbourhoods. Examples are filter operations with a certain window size.
(C) Entire neighbourhood operations considering all cells of a map as, for example, in distance calculations between cells.
(D) The neighbourhood can also be defined by a given topology as in material transport over flow networks.

We use map algebra in its extended form by iterating over discrete time steps, which allows the representation of spatio-temporal processes as in Equation (1) (c.f. Wesseling et al., 1996; Pullar, 2001, 2003; Cerveira Cordeiro et al., 2009; Mennis, 2010).

2.2. Model algebra

Map algebra enables a model builder to express environmental processes in the form of individual model components. The next step is to assemble these model components to represent an integrated system. Hence, a model builder needs to specify which model components are present in a coupled model and how these interact. By itself the provision of the model components would be sufficient for a modelling framework to determine all the interactions and the direction of the variable transfer based on the model component interfaces. However, deriving links automatically by a framework requires a formal description of model component interfaces as well as the attributes of the exchanged environmental variables. This formalisation is under development (Schmitz et al., 2012) and not included in this prototype. Here, the modeller still needs to specify which model components constitute the integrated model, which of these interact, and in which direction the variables are to be transferred. Model algebra describes the provision of the input variables $I(t)$ of Equation (1) for all model components.

Let I_l be all model component input variables $l = 1,\ldots,L$ of the integrated model. For all time steps t_i, the input variables I_l need to be provided as:

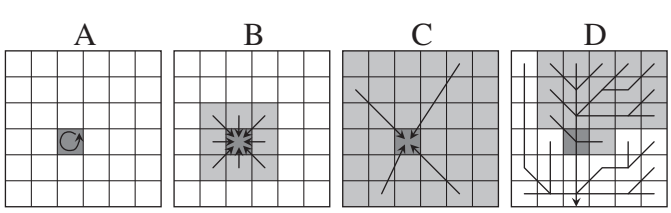

Fig. 2. Classification of map algebra operations. (A) Point or local, (B) direct neighbourhood or focal, (C) entire neighbourhood or zonal, and (D) operations on a neighbourhood given by a specific topology (Karssenberg and de Jong, 2005; Schmitz et al., 2009).

$$I_l(t_i) = A\left(E_n(t_i), O_n\left(t_{[h,i-1]}\right)\right) \tag{2}$$

where O_n denotes the output variable n that is provided by another model component of the integrated model. E_n is an external input variable n that is not a model component output such as variables obtained from a data base. A describes the transfer from either external variables or model component outputs to the input variables I_l. We consider three different situations for the functioning of A. In the first situation, the properties of the variable O_n resemble all properties of variable I_l. That is, spatial properties such as the grid extent are equal, and time steps of the model components match. In this case, A transfers the output variables $O_n(t_{i-1})$ from the previous time step directly to the model component input I_l. In the second situation, properties such as the spatial scales of O_n and I_l are different, while the time steps match. A now implies a conversion operation to adapt the properties of output $O_n(t_{i-1})$ to the properties of the input variable I_l. In the third situation, the temporal properties of the variables O_n and I_l are different: the time steps do not coincide as, for example, when coupling a daily to a monthly time step component. In this case, A describes the aggregation of a set of output variables $O_n(t_h),\ldots,O_n(t_{i-1})$ such that the properties of the input variable I_l are matched. The first two cases are not further discussed, because in these cases variables can be exchanged straightforwardly between model components which have the same temporal properties, and an adaptation of spatial properties can be realised by using existing upscaling or downscaling approaches (e.g. Blöschl and Sivapalan, 1995; Bierkens et al., 2000) before transferring the variable. For the third situation, i.e. different time steps between model components, we will demonstrate different coupling scenarios and clarify the aggregation of component outputs.

3. Component coupling

In the previous section, we introduced model components as individual building blocks of an integrated model. We will now consider the setup of multiple model components and their coupling to integrated systems. In this section, we derive the requirements for the scheduling schemes and the model components based on examples of component coupling scenarios with different temporal discretisation. First, we outline the scenario of coupling components with fixed time steps. Next, the situation of coupling components with variable time steps is described. Finally, the incorporation of components with an undetermined start and end time is described.

3.1. Fixed time steps for all components

First, we consider a situation where all components in a model have a fixed time step that is known at the model development stage. In addition to different time steps, several interactions can occur in a coupled model. Model components may not interact at all, interact in one direction, or interact bidirectionally. Each situation brings along different scheduling requirements in terms of the data exchanges. We illustrate these situations by means of two model components, while the principles also apply for multiple components.

Fig. 3 shows the case of a bidirectional coupling between a model component C_1 calculating a process such as soil water percolation and a model component C_2 calculating for example groundwater flow. C_1 runs with a fixed daily time step whereas C_2 uses a time step of one month. Therefore, data between these components should be exchanged every month. Because of the shorter time step of the soil water percolation component C_1, the

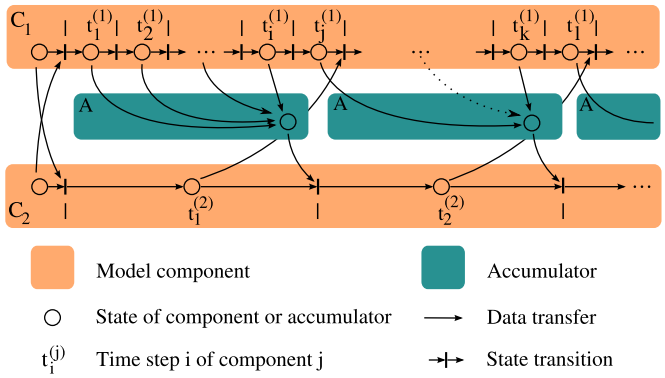

Fig. 3. Temporal control flow between interacting model components with shorter (C_1) and longer (C_2) time steps. Each model component requests the most recent output from the other component. While C_1 directly accepts the input of C_2, C_2 expects aggregated values from C_1, provided by the accumulator A. Note that data conversion issues such as spatial resampling are not explicitly included in this figure.

latest value (i.e. $t_i^{(1)}$ or $t_k^{(1)}$ in Fig. 3) is not representative for use in C_2. As a result, an aggregated value over the time steps $t_1^{(1)},\ldots,t_i^{(1)}$, such as the sum or median, needs to be calculated before it can be passed to C_2. This takes place by means of an accumulator. While most modelling situations also require an accumulator in the opposite direction, we simplify the procedure by having C_1 directly accept the latest seepage values from C_2.

In general, the order of model component execution is arbitrary as long as the dependencies are met at data exchange. The scheduling could be as follows. From its initial state, C_2 obtains data from C_1 and propagates to its new state in time step $t_1^{(2)}$. As C_2 expects new input data to proceed to its next time step that is not yet available, C_2 needs to wait until the accumulator A can provide the aggregated values obtained from C_1.

From its initial state, C_1 obtains data from C_2 and processes forward by calculating the transition function and proceeding into the state at time step $t_1^{(1)}$. Prior to the data exchange moment with C_2, C_1 can proceed independently until the data exchange moment. The state variables of C_1 need to be stored for each time step so that the accumulator can process them. C_1 can immediately continue after the data exchange moment because the output variable of C_2 is already available. C_2 can proceed to its next state in $t_2^{(2)}$ as soon as C_1 reaches $t_i^{(1)}$ and all values are processed by the accumulator.

To derive a proper order of model component execution and to determine the correct moments for data exchange, a scheduler needs to obtain information from all model components and accumulators coupled in the model. For all model components, first and last time steps and the discretisation applied need to be known. This information can be used to build up a shared timeline of all model components. Furthermore, the interaction between the components needs to be expressed, i.e. which components exchange data. From this specification and the known time steps, the data exchange moments between individual components can be derived. In case of aggregated values over time, accumulators need to be declared and executed at specific data exchange moments. The scheduler should also identify model components that do not need to wait for each other. These components can proceed independently of each other in the periods between the data exchanges and can be executed concurrently to increase the model performance.

3.2. Scheduling components with variable time steps

While a fixed time step is used in the majority of environmental model applications, not all situations can be covered with the scheduling approach outlined in the previous section. We will now consider the handling of model components with variable time steps. Modelling situations like this can occur, for example, when coupling an economic component with quarterly time step to an agricultural component for plant growth with a low temporal discretisation during the summer season and a higher discretisation during the winter season. Variations in time steps can occur in two ways: determined before the model execution, or identified during runtime.

Fig. 4 shows the situation where two model components C_1 and C_2 interact through an accumulator with component C_2 proceeding with a variable time step. In terms of the component interactions, this situation is identical to the one described in the previous section. The model component C_2 again requires aggregated output values from C_1, while the output value from C_2 can be used directly. The model component C_2 changes its time step after time step $t_1^{(2)}$ to a shorter duration. Consequently, the scheduler needs to adapt the interval considered by the accumulator as well as the following data exchange moment between the model components C_2 and C_1.

As in the previous section, the model components and accumulators as well as their interactions need to be specified to derive a schedule for the model execution. As the discretisation of time steps for each model component is not necessarily known before the model run, it is not possible to derive an execution schedule for the total runtime in advance. Therefore, it is necessary to evaluate the progress of the model components and to derive the corresponding scheduling during runtime. By querying each model component for the end of its current time step, the next data exchange moment between model components can be determined. Using this information, a schedule can be generated and the model components can be executed. When the components reach the data exchange moment the time steps of the model components need to be evaluated again. This alternating scheme of component execution and schedule generation continues until the end of the simulation.

3.3. Scheduling components with unknown start and end time

The implications of the existence of components with variable time steps on the prediction of the execution scheme were already discussed in the previous section. We now consider the case where even less knowledge about the component lifetimes is available: a situation in which an unknown, limited number of model components can appear and disappear during model runtime (see Fig. 5) as in the case of agent-based models. An example is an integrated model where a field-based groundwater component is coupled to a

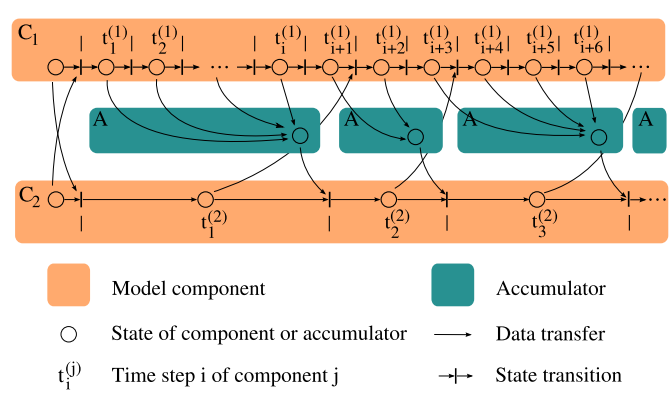

Fig. 4. Temporal control flow between two interacting components where model component C_2 changes the time step duration. The scheduler needs to update the interval considered by the accumulator and to notify C_1 of the advanced availability of output variables from C_2.

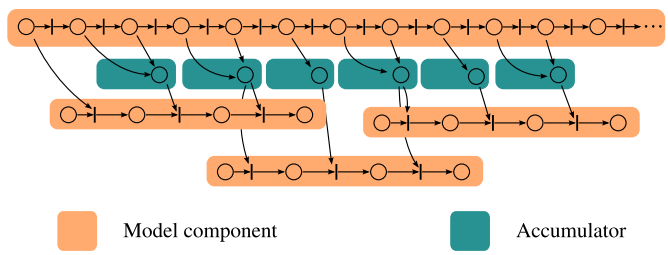

Fig. 5. Components with limited life time interact with a component with continuous time steps. The scheduling needs to update the interval considered by the accumulator and to keep track of the number of individual components. Also, aggregated values are used as initial values for the individual components.

number of individual model components representing individual trees. The trees have their own confined lifetime, can spawn offspring, and be impacted by water extraction from the groundwater component. Before the simulation run, the functions and parameters describing these processes are known. In addition, the initial population with random age values is known, while the variation in the number of trees is unknown before the model run.

To organise the scheduling of components with unknown lifetime, the scheduling needs to be arranged in a flexible way that responds to the existing model situation and the number of participating components. Therefore, the scheduler can only consider a short time frame in advance. To achieve this, the components need to be registered at the scheduler as well as the accumulator used to communicate with other components. This registration allows the scheduler to generate dynamic lists that hold the properties of those individual components. These lists are continuously updated and track modifications of current components (such as end of lifetime) or the appending of new components. With these dynamically managed lists of components, queries about components can be executed during model runtime.

The interactive components must hold their start time, end time and time step that determine the temporal properties of the model component. As the number of components is unknown during runtime, additional effort in the communication between the components and the scheduler is required. For example, a tree model component calls the scheduler to obtain the neighbouring trees at specific moments in time. In addition, a component needs to obtain information about the scheduler to forward this information to the spawned component allowing its self-registration at the scheduler. It is recommended to construct autonomous components that can be activated by and communicate with a central control instance. This enables a scheduler to spawn components at individual moments during the model run in a flexible way.

3.4. Accumulators

When a model builder intends to couple newly constructed model components, or model components taken from other model builders or from a library of preconstructed components, the situation may arise where individual components operate on different time steps. An orchestration of these individual model components therefore results in a set of different time steps. Resolving the different discretisations is necessary to couple model components appropriately.

A potential solution is to modify the individual model components such that they share a common time step. Here, two options can be distinguished. One is that all model components are forced to the time step of the model component with the longest time step. However, model components may have an upper boundary for their time step due to the dynamics represented either by the model component itself, or due to limitations with numerical solutions, so this solution is not feasible. The other solution is to force all model components to the time step of the component with the smallest time step. To conform to the smaller time step, however, an additional overhead would be introduced by a repeated execution of a model component with a longer time step. In addition to these problems, the modification of a model component to match a common time step contradicts the principle of independent model components. By adapting the time step, a model component would not only reflect interaction with other components but also a temporal dependency imposed by other model components. These temporal dependencies make a model component less generic and therefore less reusable for coupling in a different modelling context. It is therefore not desirable to use a common time step in coupled models.

To allow for a coupling of model components while retaining their individual time steps, the modelling framework needs to include accumulators connecting model components. In this approach, the accumulators obtain a set of values at the input interfaces, aggregate with a certain operation, and provide the result at the output interface. We propose a predefined set of generic accumulators that is capable of time aggregation for a large range of system variables. Several operations can be used to calculate the aggregated values. Their classification based on the operation type is shown in Table 1. These generic accumulators provide point operations executed on a temporal interval. Fig. 6 illustrates the working of an accumulator, in this case accumulating Boolean maps with the `OR` operation. Values from the component with smaller time steps $t_i^{(1)}$ are obtained for the time interval since the last exchange moment. For each cell location, the operation $g_{x,y}$ aggregates the input values and assigns the values to a result map. This map is provided at the output interface and ready for collection by the component with larger time step $t_j^{(2)}$.

Next to aggregating variables over time, modelling situations can occur where model component attributes need to be adapted. An example is a difference in spatial discretisations requiring a resampling of the grid cell size, or a unit conversion as in translating Fahrenheit to Celsius. These generic conversions follow the structural adapter pattern (e.g. Gamma et al., 1995) to enable efficient and adequate coupling of model components.

Many modelling scenarios can be built with generic accumulators as described above. However, coupling scenarios can appear where more complex aggregation operations are required such as in moving window operations with user specific time step ranges. The model builder can be supported with this task by extending the accumulator templates to implement the processes according to the desired purpose.

4. Technical implementation

Following the concepts described in the previous section, we developed a prototype framework consisting of two layers (Fig. 7). The layer for custom model development provides the modeller with the instruments to describe and design model components

Table 1
Classification of generic accumulator types that aggregate a time series of raster maps.

Type	Operation g, e.g.	Variables
Logical	`AND, OR`	Truth values that describe epidemic plague or fire occurrences
Arithmetic	`sum, mean`	Average temperature or total precipitation
Conditional	`min, max`	Lowest particle concentration values or peak flow values exceeding a threshold

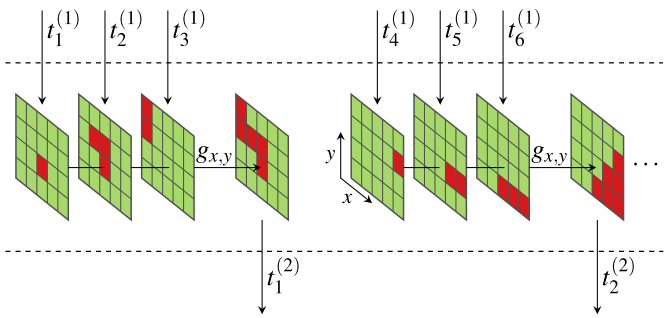

Fig. 6. Schematic workflow of an accumulator connecting two components with different time steps $t_i^{(1)}$ and $t_j^{(2)}$. The operation g, here the Boolean OR, aggregates the set of input maps obtained since the last exchange moment. Cell values are coloured in red for `True` and green for `False`. Left to right: time; x and y represent spatial coordinates. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

and coupled models. A second layer provides the execution management for schedule generation, model execution and data transfer. The concepts introduced in this section are universal and the implementation is in principle independent of the chosen programming language. In our prototype, we provide the modeller with a toolbox accessible from the Python (2013) language, which we will introduce first. Then, we present the implementation details of the layers for custom model development and execution.

4.1. Python as a basis for a modelling environment

Many phenomena in human–natural systems are currently being studied in an integrated way. Integrating different domains likely requires merging a broad range of technical knowledge and computational backgrounds. Environmental modellers, such as hydrologists or ecologists, who perform the integration, now face the situation that they may be inexperienced in a certain programming language or, if they are proficient, they may be inexperienced in a domain that needs to be integrated. It is advantageous if a modeller can use a modelling environment that requires a limited amount of training (Karssenberg, 2002). In addition, a software environment should preferably support an exploratory model construction workflow (e.g. De Kok et al., 2010) and ease the communication with other scientists and non-modellers (c.f. Fall and Fall, 2001). A declarative development language often meets these demands (De Kok et al., 2010). Finally, it is beneficial if a modelling environment can be extended to handle functionality not foreseen by software engineers and modelling framework developers.

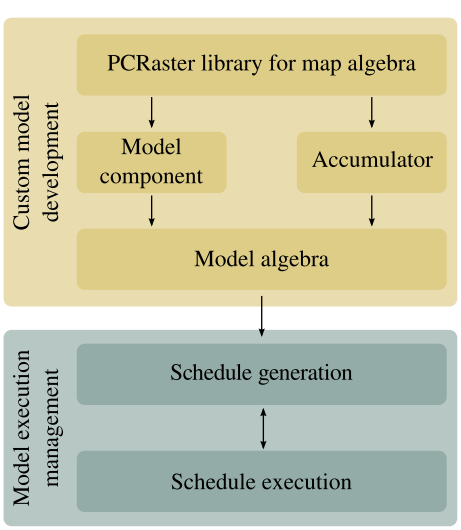

Fig. 7. Architecture of the modelling framework. The custom model development layer provides framework functionality for component and model building. The model execution layer comprises of the schedule generation and execution management.

We use the interpreted, high-level programming language Python (2013) as a foundation for our modelling environment. Python has been recognised as the de facto standard for exploratory, interactive, and computation-driven scientific research (Millman and Aivazis, 2011; Lin, 2012). The scripting language provides a readable and concise syntax, making it accessible to scientists that are not programmers (Bäcker, 2007; Pérez et al., 2011). Python supports multiple programming paradigms such as object-oriented, imperative, and functional programming styles allowing users to choose the paradigm that is appropriate for their particular problem. Python as a platform supports various steps in the scientific workflow in addition to the computational parts, such as data preprocessing, access to web services, or distributed computing (e.g. Oliphant, 2007; Best et al., 2007; Langtangen, 2007). A wide range of scientific data formats (e.g. GDAL Development Team, 2013; OGR, 2013; HDF5, 2010; XML, 2008), and analysis or visualisation tools (e.g. RPy, 2011; Hunter, 2007; Schroeder et al., 2000) are supported as well. Python is also used as an embedded modelling language in commercial and open-source applications such as ArcGIS (2013) or QuantumGIS (2013).

The natural syntax allows a straightforward use of the arithmetical and array data types provided by the Python standard library. However, not all native data types match the level of thinking of scientists such as hydrologists or ecologists, which is a potential disadvantage as it is preferable that modellers can use data types and operations that represent domain concepts (Karssenberg, 2002). One solution is to add these domain specific concepts to the language. For spatio-temporal modelling, spatially distributed attributes such as land use types or hydrological characteristics can adequately be represented by fields. Fields allow for the continuous representation of particular properties and a good algorithmic analysis by means of map algebra (Tomlin, 1990). Discrete time systems are appropriate to express spatial dynamics because the dynamic changes can be expressed using a stepwise model of execution over discrete time and spatial intervals (Pullar, 2003). Karssenberg et al. (2007, 2010) introduce Python modules that enable modellers to use map algebra operations for the construction of dynamic and stochastic models.

A potential disadvantage of Python as an interpreted language, however, is the focus on programming productivity rather than on execution performance. Pure Python implementations of array operations are considerably slower than comparable constructs in system programming languages (e.g. Langtangen, 2007). However, optimised modules for scientific computing (SciPy, 2013; Van Der Walt et al., 2011) compensate for limited standard performance. Moreover, the mixed-language approach provided by Python allows to integrate Python and system-programming languages such as Fortran, C and C++. By using bindings such as Cython (Behnel et al., 2011), Boost.Python (2012), or F2PY (Peterson, 2009), performance critical parts can be executed in system languages while a modeller uses the corresponding high-level operations in Python. The language bindings also allow extending the Python environment with legacy code developed in system programming languages.

4.2. Custom model development

4.2.1. Map algebra

We utilise the PCRaster modelling environment (Wesseling et al., 1996) and its map algebra implementation as a framework

for modellers to implement spatio-temporal processes of model components. PCRaster provides the management and visualisation of raster-based two-dimensional maps and three-dimensional block structures (Karssenberg and de Jong, 2005; Pebesma et al., 2007). A wide range of operations implementing spatial and temporal algorithms on these data types is included in the modelling environment. These algorithms are implemented in C++ and made available as a Python module by using the Boost.Python language binding library (Boost.Python, 2012). This allows a model builder to describe a model component in a high-level scripting language while the computational intensive map algebra calculations on spatial variables retain the performance of a system programming language.

A model builder can use and combine these operations to construct the transition function *f* from Equation (1). For instance, the `windowtotal` operation from the PCRaster module implements a direct neighbourhood operation (Fig. 2B):

```
burningNeighbours = windowtotal(fire, 3)
```

The `windowtotal` operation receives two input arguments. The first argument `fire` is a Boolean map indicating if a cell is on fire or not, the second argument is a map containing the number of cells defining the window size in horizontal and vertical direction for each grid cell. A uniform value of 3 is used, because the window size is uniform in space. The operation returns a result map where each cell obtains the sum of values in a square neighbourhood. All map algebra calculations operate on raster maps and return raster maps.

Further information about the development of PCRaster and its Python bindings can be found in Karssenberg et al. (2007). Recent applications of the PCRaster modelling environment can be found, for instance, in stream flow prediction at catchment scale (e.g. Zhao et al., 2011), flood forecasting for transnational European river basins (e.g. van der Knijff et al., 2010); assimilation of atmospheric transport models (e.g. Hiemstra et al., 2011); uncertainty estimation in land use change modelling (e.g. Verstegen et al., 2012); or calculation of surface water availability (e.g. van Beek et al., 2011) and the groundwater footprint (e.g. Gleeson et al., 2012) at global scale.

4.2.2. Framework implementation

Within the layer for the custom model development (Fig. 7), the framework provides templates to allow for the development of model components and the use of accumulators, and their coupling to integrated models. Fig. 8 shows the Unified Modelling Language diagrams (Booch et al., 2005) of the three base classes and their associated methods that are provided to the model builder. The `ModelComponent` class provides template functionality for the building of model components and can be completed by the modeller with map algebra instructions. The `Accumulator` and `CoupledModel` classes provide functionality for the model algebra operations that can be used to specify a coupled model setup.

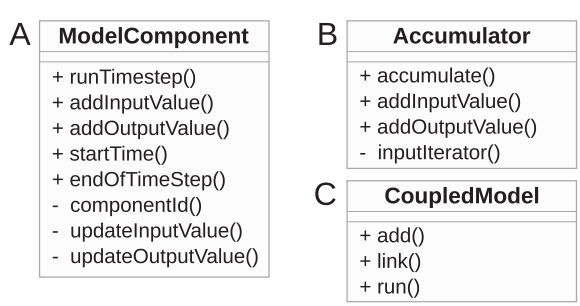

Fig. 8. UML diagram of the classes provided by the framework for model development. ModelComponent (A) needs to be filled with functionality by the model builder. Accumulator (B) and CoupledModel (C) provide methods to specify the integrated model.

The diagram of Fig. 8A shows the class methods that are available to build model components. The modeller implements the `runTimestep` method by inserting map algebra operations describing spatial processes. The modeller can use the remaining public methods to describe component attributes. As indicated in the previous section, model components need to provide a specified external interface for their input and output variables, and a means to formulate the process description for the time step proceeding function. With the `addInputValue` and `addOutputValue` methods, variables can be declared as an input or as an output variable, respectively. The temporal extent of the component is specified by two methods. In the `startTime` method, the modeller needs to provide the starting time of the model component. In the `endOfTimestep` method the end of the current time step is given. A modeller can use the default implementation of `endOfTimestep` returning a constant time step specified at the initialisation of a model component, or the modeller can override the method to return changing time steps, for example, by implementing the leap year handling required for a yearly time step. The execution layer will repeatedly call the `endOfTimestep` method to obtain the current modelling time step of the component. As a consequence, it is possible to change the time step duration at runtime.

The model component base class also holds private methods that are not meant to be used by the model builder. These methods provide functionality used by the framework mainly for accounting and obtaining runtime information. Methods such as `componentId` return a unique identifier required to address model components during model execution. The `updateInputValue` and `updateOutputValue` methods are called by the framework at the data exchange moments. These methods realise the transfer of input and output variables between model components and accumulators.

Within the framework, accumulators are used to collect results of a specific component, to execute an aggregating operation and to pass the processed data to a connected component. Accumulators enable environmental modellers to implement functionality comparable to aggregation functions in databases or reduction functions in programming languages. In Fig. 8B the base class and its methods for the accumulator development is shown. When none of the generic accumulators can be used by the model builder, a specific aggregation operation can be implemented within the `accumulate` method. The `addInputValue` method specifies the variables received from other components that need to be aggregated. During model runtime, the input values taken from a model component with smaller time steps are collected and stored. With the help of an iterator method, the model builder can obtain and process the input data since the last data exchange moment. The `addOutputValue` method specifies the variable that provides access to the aggregated values for other components.

Fig. 8C shows the class for the specification of a coupled model. The class provides two methods to specify the information about the model components, accumulators and their interactions. The model builder is able to `add` model components and accumulators to the coupled model and to specify interactions between the components and accumulators with the `link` method connecting output to input variables. The `run` method executes the coupled model.

4.3. Model execution management

The coupling scenarios described in the previous section require the handling of model components with various time steps and the consideration of accumulators for temporal aggregation to ensure an ordered execution of the coupled model. We now describe the algorithm that manages the schedule generation and execution.

The ordered execution of individual processes is required in other disciplines as well and can be found, for example, in the process scheduling of operating systems (e.g. Bach, 1986; Tanenbaum, 1992; Torrey et al., 2007) or production line planning executed in operations research (e.g. Koomsap et al., 2005). Inter-component communication to notify one or more components about the status and requests for data can be realised by different approaches. In a pull-based approach, model components request data from other components and thereby initiate the progress of the delivering components. This approach is followed, for example, in OpenMI version 1.4 (Moore and Tindall, 2005). A second approach is to keep a centralised instance organising the execution and communication of model components as for instance realised in the Tarsier framework (Watson and Rahman, 2004).

Here we use the second technique by implementing a client-server approach. By maintaining information about all model components, accumulators and their temporal status, one scheme can be applied to generate the schedule for the execution of the integrated model. Also, a central coordination of the execution of the components enables modellers to add and remove model components at runtime which is required for agent-based modelling. Furthermore, we can identify independent model components and initiate their concurrent execution to increase the overall runtime performance. The additional administrative overhead of a central instance is marginal, as maintaining and updating time step information is inexpensive compared to the computational intensive spatial processes executed in the model components.

The modelling framework provides a centralised instance that arranges the management of the coupled model. The instance builds up a shared timeline between components and accumulators and generates a schedule by accounting for the current time steps of the model components during the runtime. Table 2 shows the execution algorithm of the schedule generation scheme for a single run of an integrated model. The input information for the algorithm is the set of the model components and the set of accumulators. The model components also need to specify their start time and initial time step. Moreover, the interactions between model components and accumulators need to be specified.

An integrated model is then executed as follows. During the initialisation of the model run, accounting lists for model components and accumulators are created. These lists are filled with parameters obtained from the model components such as the component identifier and the starting time steps. The lists are also enriched with additional information for runtime management like status flags indicating if a component, for example, either needs to wait or is able to proceed its time step. These lists are updated continuously during the model run and form the basis to create sets of components grouped according to their status flags. For example, at the start of the integrated model the runnable set contains all model components. The initial values are obtained from the coupled components and then the first time step is executed (lines 1–4).

For the consecutive time steps, the algorithm is executed continuously until all components have been completed. In a first phase, the status of the model components is determined (lines 6–14). Then, the execution of model components, accumulators and the data transfer is managed. The end of the time step is obtained from each waiting component to determine the set of runnable model components (line 6). This information, in combination with the specified interactions, is used to determine if a model component needs to interact with one or more components within its next time step. If this is not the case, the component status can be set to runnable (lines 8–9). Otherwise, the progress of the coupled model components needs to be assessed. Therefore, for each link the temporal progress of the providing model component is evaluated. If the time steps of all linked components exceed the time step of the model component in question, the most recent input data is available and the component is able to proceed (line 12). Otherwise, the model component execution is delayed until all data from the coupled components is available.

Once the model components have been assessed and grouped into waiting and runnable sets, the execution and the transfer of variables needs to be arranged (lines 15–17). Before the model components with the status runnable can progress their time step, the output data needs to be distributed to the input variables of connected components. The scheduler obtains data from the output variables, calls the accumulators to aggregate data where appropriate, and distributes data to the input variables. Finally, with the input requirements met to proceed to its next time step, a model component is scheduled for execution.

Table 2
Simplified algorithm for a single run of an integrated model.

```
1   initialise data structures
2   build set of runnable model components
3   distribute intial data
4   proceed first timestep of model components
5   while components not finished do
6     obtain end of time step for all components
7     for each component MC do
8       if MC has no links then
9         add MC to set of runnable
10      else
11        if all coupled components exceed time step of MC
          then
12        add MC to set of runnable
13        else
14        add MC to set of waiting
15    execute accumulators
16    distribute output data to input variables
17    execute all components with status runnable
```

5. Case study

In the case study, we show how the framework can be applied to construct and couple model components. We will demonstrate this by developing a biomass growth model which is coupled to a fire-spreading model. With these two model components, we link the contrasting processes of biomass growth and biomass destruction due to fire. These processes operate on different time steps. Further on, we demonstrate re-usability and extensibility by modifying the biomass component with a more complex process description, and add another model component simulating biomass removal generated by human activities such as logging.

The example is only offered for demonstration purposes and should not be considered as a reliable modelling solution for the described problem. We focus on describing the integration of model components with hourly, daily, or yearly time steps and omit essential aspects typical for this type of models. We simplify the process descriptions by assuming linear or logistic growth functions for the biomass component, fire spread on available biomass and topography with limited fire duration, and increasing harvest costs as a function of the distance from roads.

5.1. Coupling of vegetation biomass growth, wildfire and harvesting model components

First, we consider a bidirectional data exchange between two model components using different temporal discretisations. A biomass growth component runs with a fixed time step of one year. A fire model component evaluates once per day the chance that a fire occurs. In case of a fire, the time step changes to one hour until

the fire extinguishes. The model is applied to a catchment in the Spanish Pyrenees. The study area has a minimum elevation of 913 m, a maximum elevation of 1338 m, and consists of 27,925 cells with a cell length of 10 m.

5.1.1. The biomass growth model component

This model component calculates the amount of biomass for each cell in the catchment. For demonstration purposes, we begin with a linear growth process. The model component provides the available biomass as output variable to other model components. The input variables consist of the cells that were burned in the last year, and the biomass removed by human influence. Table 3 shows the implementation of the component. The component simulates over a 50 year time span with a yearly time step starting from January 1, 2000 (line 3). Lines 4–9 specify the input and output interface. The initial biomass is read from a stored data set containing random biomass values distributed over the catchment, and afterwards specified as a component output (lines 4 and 5). The burned area is set to an initial value of zero and defined as component input (lines 6 and 7). This variable will be updated each year with values from the fire model component. Lines 8 and 9 set an initial value of harvested biomass and define this variable as second component input. As the variable will not be provided by another component here, the value remains constant during the model run.

The processes described in the `runTimestep` (line 11) method are calculated for each time step. Only the cells affected by a fire are evaluated by resetting the biomass of the corresponding cells (line 12). The amount of biomass per cell is increased by a constant growth value of 25 and diminished by the harvested biomass (line 13). The growth is assumed spatially homogeneous, while spatially variable growth is feasible with minor modifications. Finally, the biomass variable is written to disk with the `report` operation (line 14).

The operations declared in the model component act on two-dimensional raster maps. Fig. 9 shows how these maps can be examined with the visualisation tool Aguila (Pebesma et al., 2007). The left window shows the spatial distribution of the biomass in the catchment, and the graph on the right shows the biomass growth over time for the grid cell under the crosshair.

5.1.2. The fire model component

The second model component simulates wildfires. Wildfires are complex phenomena and rely on variable conditions, such as the wind velocity. We simplify by considering the spread of fire relying on two conditions. First, the spreading of fire depends on the available biomass in the neighbouring cells. This input data is provided by the biomass model component. Second, the spread of fire depends on the topography of the catchment with a higher probability of burning in uphill direction (c.f. Karafyllidis and Thanailakis, 1997). The output of the component is a map with Boolean values indicating which cells of the catchment were affected by the fire.

The corresponding implementation of the fire model component is shown in Table 4. Similar to the biomass component, the simulation period and the input and output variables are initialised in lines 3 to 7. The fire spreading processes given in the `runTimestep` method are calculated if a new fire starts or an existing fire continues to burn. This condition is fulfilled when either a random value exceeds a certain ignition threshold, or a fire already started in one of the preceding time steps (line 10).

If a new fire ignites, a random starting location is assigned and the time step is changed to one hour (lines 11–13). Then, the cells that potentially catch fire are calculated by identifying the non-burning neighbour cells with available biomass above the threshold (lines 14–18). For these cells, a higher probability of ignition is assigned to uphill cells. The uphill cells are determined based on the local drain direction network map `self.ldd` (c.f. Burrough and McDonnell, 1998) and fire occurring in the neighbouring downstream cell (lines 19–23).

The lines 24–26 account for the fire duration and extinction for each cell. We assume that a fire in a cell extinguishes after 10 h (line 26). In line 27, the total area that is burned within this fire is updated. If no cell in the catchment is burning anymore, the time step is reset to one day (line 28 and 29).

5.1.3. Aggregation of fire incidents

As the time step of the fire component is smaller than the time step of the biomass growth component, several individual fires can occur within one year. To account for the total burned area it is necessary to aggregate the cells affected in the individual fires before the data is processed by the biomass component. Therefore, an accumulator interconnects the biomass and fire model components.

Similar to model components, accumulators are equipped with an external interface to define the input and output variables. The burned areas provided by the fire component are collected as input values during the model execution and aggregated for each year. The input maps of the accumulator hold Boolean `True` values in the cells that were burned during a fire. The accumulator iterates over the individual input values and aggregates in this case with a Boolean `OR` operation, as was shown in Fig. 2. The resulting output variable holds the total burned area of the current year.

5.1.4. Component coupling and results

The model algebra script that is used by the modeller to implement the coupled model is shown in Table 5. Instances of the model and the two components are created in the lines 1–3. In line 5 and 6, the model components are added. The transfer of data that does not require an accumulator is specified in line 8, where the output variable `biomass` from the biomass component is linked to the input variable `availableBiomass` of the fire component. Line 9 specifies the link in the opposite direction. By a logical `OR` accumulator, the output variable `burnedArea` of the fire component is aggregated and then transferred to the input variable `burned` of the biomass component. Finally, the complete model is executed in line 11.

5.1.5. Replacing the biomass component

During the development of an integrated model, modifications to the process descriptions emerge because of, for example, the increase of scientific knowledge, increase in data availability, or

Table 3
Python script showing the linear growth process simulated in the biomass component. The variables hold two-dimensional raster data types.

```
1   class Biomass(PCRasterRealTimeComponent):
2     def __init__(self):
3       PCRasterRealTimeComponent.__init__(self, "clone.map", ↩
    datetime(2000,1,1), datetime(2050,1,1), timedelta(days = 365))
4       self.biomass = self.readmap("initialBiomass")
5       self.addOutputValue(self.biomass)
6       self.burned = boolean(0)
7       self.addInputValue(self.burned)
8       self.harvested = scalar(20)
9       self.addInputValue(self.harvested)
10
11    def runTimestep(self):
12      self.biomass = ifthenelse(self.burned, 0.1, self.biomass)
13      self.biomass = self.biomass + 25 − self.harvested
14      self.report(self.biomass, "availableBiomass")
```

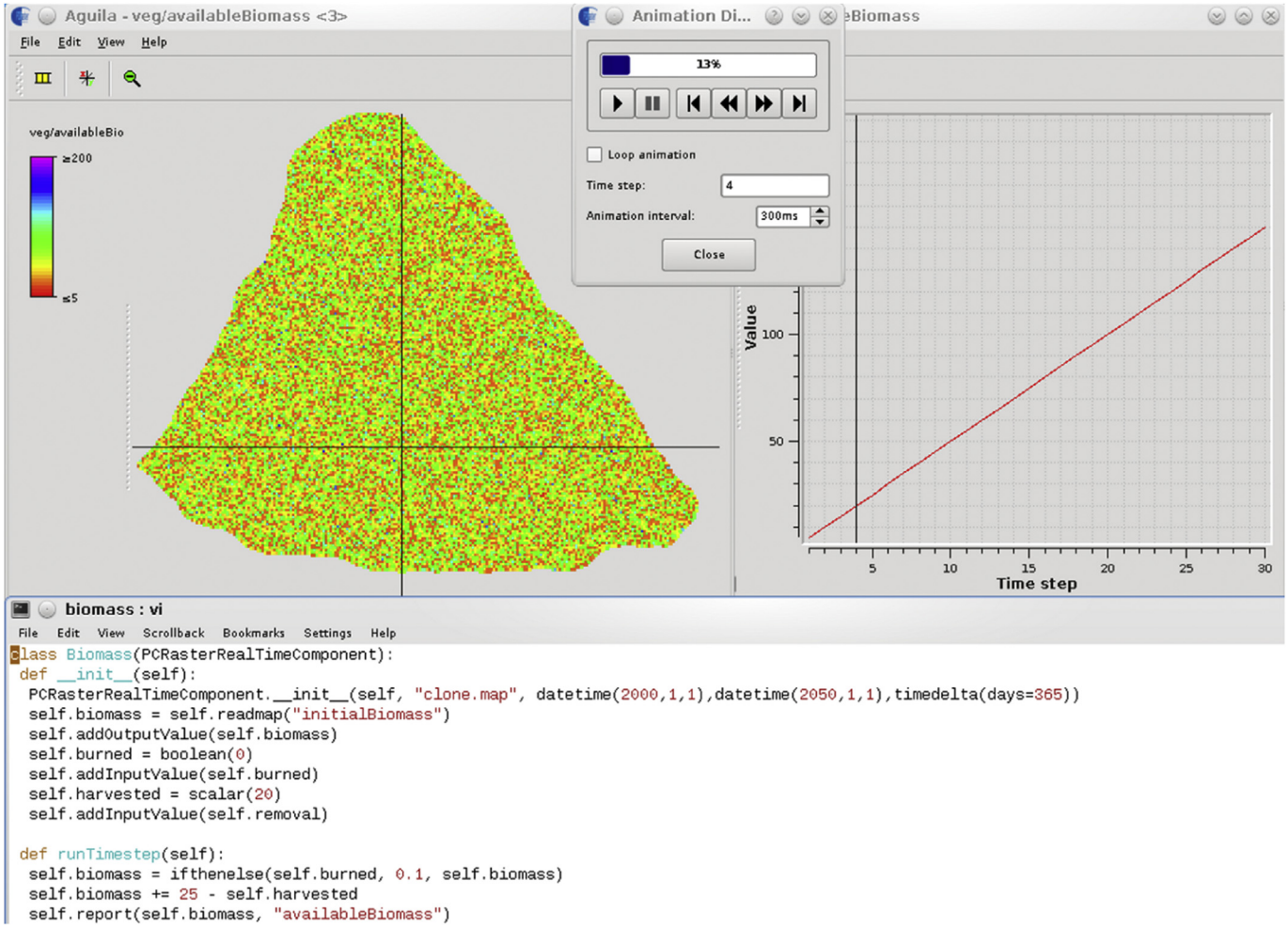

Fig. 9. Screen capture showing the spatial distribution of biomass and the timeseries for a specific cell. Cell locations and instants of time can be selected interactively.

model component maintenance by activities like bug fixing or refactoring. We now demonstrate how a specific component can be adapted without affecting the other components or the overall working of the coupled model. Therefore, we replace the simplified linear growth of the biomass B by a logistic growth function (c.f. May, 1977; Dakos et al., 2009):

$$\frac{dB}{dt} = rB\left(1 - \frac{B}{k}\right) - (c_0 + c_{inc}t)\frac{B^2}{B^2+1} + dD_{x,y} + \epsilon \tag{3}$$

The script in Table 6 shows the more sophisticated biomass component, where the first ten lines are the same as in Table 3. Equation (3) is calculated in the `runTimestep` method as follows. For simplicity, we set the initial grazing pressure c_0, its increase $c_{inc}$and the random noise ϵ to 0. Line 13 calculates the growth term with a growth rate r of 0.2 and a carrying capacity k of 100. The diffusion of the biomass is composed of an assumed dispersion rate d with a value of 0.01 and the dispersion term $D_{x,y}$ (line 16). $D_{x,y}$ models the interactivity with the direct neighbouring cells representing clonal growth or seed dispersal. The neighbouring cells are obtained with the `window4total` operation. Finally, the biomass variable is updated and stored (lines 18–19).

Fig. 10 shows resulting maps for both components and the accumulator for a 50 year model run. The top row presents output maps for the biomass component, every 5th year is shown. The bright spots indicate areas that were burned previously. The bottom row shows four output maps from the fire component resulting from a fire incident occurring in August 2016. The bottom right map shows the total burned area accumulated in the year 2016.

5.1.6. Adding a human interaction component

In addition to the modification of individual components, it is often desirable to insert new model components to implement additional processes. We now add a third component simulating biomass removal by human activity, and replace the constant parameter `harvested` of the model script in Table 3 with the output of the new model component `Logging`. The input variable for the model component is the biomass available in the catchment. The output variable is the harvested biomass.

The initialisation of the model component is similar to the ones previously presented, the `runTimestep` implementation is shown in Table 7. We assume that the costs of logging increase with the distance from the infrastructure present in the catchment, and that a maximum of feasible logging costs is given. These conditions form a zone of acceptable logging costs along the roads (see Fig. 11). Within this zone, 20% of the available biomass above a certain biomass threshold is harvested (lines 11–14). The remaining lines 15–17 calculate total values for the costs and the logged biomass.

To the script of Table 5, we append instantiation and adding of the `Logging` component to the model, and specify the links for the bidirectional data exchange between the `Logging` and `Biomass`

Table 4
Python script showing the implementation of the fire model component. The initialisation of some variables is omitted. The literals &, | and ~ represent the spatial Boolean AND, OR and NOT operations.

```
class Fire(PCRasterRealTimeComponent):
  def __init__(self):
    PCRasterRealTimeComponent.__init__(self, "clone.map",
    datetime(2000,1,1), datetime(2050,1,1), timedelta(days = 1))
    self.availableBiomass = scalar(0)
    self.burnedArea = boolean(0)
    self.addInputValue(self.availableBiomass)
    self.addOutputValue(self.burnedArea)

  def runTimestep(self):
    if random.random() > self.ignitionThreshold or
    self.fireIsBurning == True:
      if self.fireIsBurning == False:
      self.assignFireLocation()
      self.setTimeStep(timedelta(hours=1))
      nrBurningNeigh = window4total(self.burning)
      burningNeigh = ifthenelse(nrBurningNeigh > 0,
      boolean(1), boolean(0))
      potentialNewFire = burningNeigh & ~self.burning
      bioNeigh = ifthenelse((potentialNewFire == 1) &
      (self.availableBiomass > 30), self.availableBiomass, scalar(0))
      bioAndFire = self.burning | boolean(bioNeigh)
      neighboursToBurn = windowtotal(bioAndFire,
      3) − scalar(self.burning)
      potentialNewFire = ifthenelse((neighboursToBurn >
      1) & (bioNeigh > 0), potentialNewFire, boolean(0))
      downhillBurning = downstream(self.ldd, self.burning)
      prob = ifthenelse(downhillBurning, scalar(0.8), scalar(0.1))
      newFire = (uniform(1) < prob) & potentialNewFire
      self.burnsSince = ifthenelse(self.burning, self.burnsSince + 1,
      self.burnsSince)
      self.burning = self.burning | newFire
      self.burning = ifthenelse(self.burnsSince > 10, boolean(0),
      self.burning)
      self.burnedArea = ifthenelse(self.burning, boolean(1),
      self.burnedArea)
      if self.nrBurningCells(self.burning) == 0:
      self.setTimeStep(timedelta(days = 1))
```

components. Fig. 11 shows resulting maps obtained with the extended model. In the output of the biomass component the influence of the logging constrained by the maximum logging costs is visible (Fig. 11). Also, the fire in the south-west of the catchment reduces the area of biomass available for logging (Fig. 11B). Fig. 11C shows the influence of the reduced biomass on the fire spread. The total costs and the amount of harvested biomass can be determined because the costs and amounts of logged biomass are stored for each cell and each time step in the logging component (Fig. 11D).

5.1.7. Collaborative building of larger integrated models

Above, we showed how modellers can construct and modify model components for defined spatio-temporal processes, and how they can couple and extend integrated models. The construction of integrated models with a larger number of components, however, will require additional effort to integrate external models and model components constructed by other scientists.

Existing, external models can be integrated following the wrapper approach. An external model can thereby be encapsulated in a model component and called per time step within the `runTimestep` method, based on two different practices. The first approach can be used with compiled executables that have a command line interface. Here, the inputs and outputs of the model components need to be converted to the file format used by the executable, and its execution can be accomplished by a system call (c.f. Carrera-Hernández and Gaskin, 2006; Schmitz et al., 2009). The second approach is applicable if the source code of the external model is available. In this case, the language binding approaches introduced in Section 4.1 can be used avoiding the detour via the file system (e.g. Lin, 2009). The framework supports the modeller in the construction tasks by providing predefined building blocks, and it provides a basic inspection of linkages based on the intrinsic data types of Python and PCRaster.

Collaborative development of integrated models places further demands on the modelling framework. Additional information clarifying the scientific intent of model components is required to improve the understanding and to ease the reuse of available model components. Moreover, cooperation between modellers of various disciplines requires a coordinated development workflow. In the current version of the framework, the model component interfaces are described at the technical level of the application programming interfaces (APIs) of the Python language. However, APIs make it difficult for scientists to detect conceptual mismatches when linking models across scales (Ewert et al., 2009) and scientific domains, as APIs do not always expose semantics such as defining a model variable as *parameter* with an associated measurement unit. A possible solution is to use ontologies (Uschold and Gruninger, 1996) expressing the semantics of model components, including for instance the exchanged variables, constraints of model components, and interfaces. Ontologies can improve the assessment of a model setup as they allow incorporating additional information such as measurement units and spatial discretisation in the evaluation of the linkages. They also can lead to implementation independent representations and an improved description of model components or integrated models. Ontologies are used, for example, to describe environmental data (e.g. Madin et al., 2007; Saiful Islam and Piasecki, 2006), or to describe component interfaces (e.g. Rizzoli et al., 2008; Athanasiadis et al., 2011), including component models used as building blocks in the framework described here (Schmitz et al., 2012).

Managing the collaborative model development is another facet of the integrated modelling process. Applying common model building practices is required to coordinate a distributed

Table 5
Model algebra script specifying the coupled model consisting of the two model components, a logical accumulator, and their interactions.

```
biomassFire = CoupledModel()
biomass = Biomass()
fire = Fire()

biomassFire += biomass
biomassFire += fire

biomassFire.link(biomass["biomass"], fire["availableBiomass"])
biomassFire.link(fire["burnedArea"], biomass["burned"], OR)

scheduler.SingleRun(biomassFire).run()
```

Table 6
Python script for the improved biomass component. The input and output interface to other components remains the same, only the process description is modified.

```
def runTimestep(self):
  self.biomass = ifthenelse(self.burned, 0.1, self.biomass)
  growth = 0.2* self.biomass* (1 − self.biomass / 100.0) −
  (self.biomass**2 / (self.biomass**2 + 1))
  sumBiomassOfNeighbours = window4total(self.biomass)
  numberOfNeighbours = window4total(spatial(scalar(1)))
  diffusion = 0.01* (sumBiomassOfNeighbours −
  numberOfNeighbours* self.biomass)
  growth = growth + diffusion
  self.biomass = self.biomass + growth
  self.report(self.biomass, "availableBiomass")
```

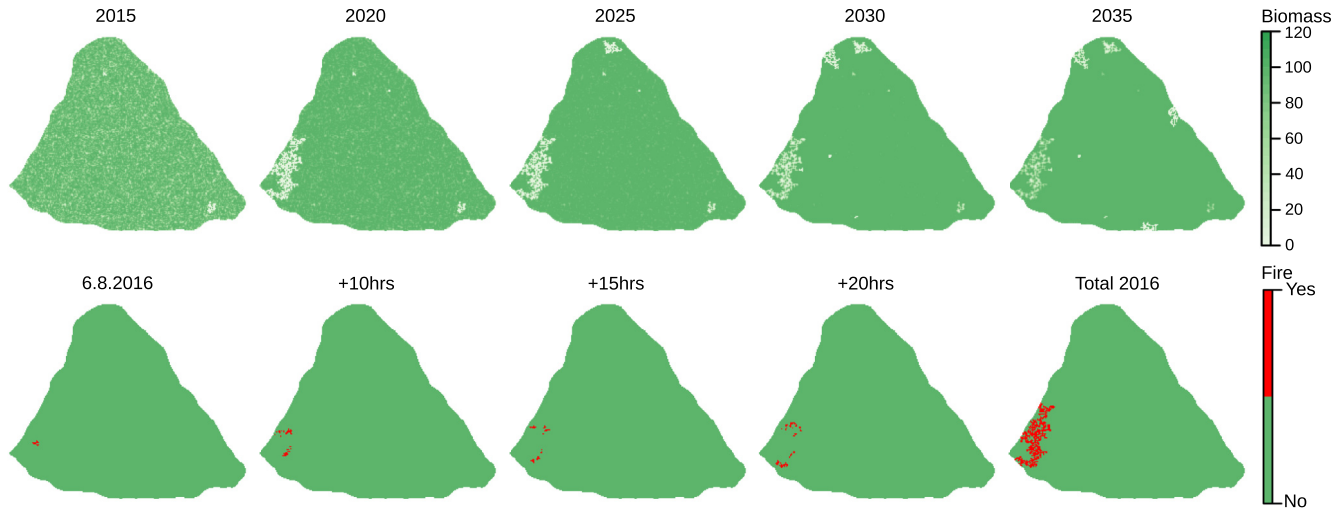

Fig. 10. Result maps from a model run with the improved biomass component. The top row shows biomass values, the bottom row output of the fire component and the accumulator.

development of modular components and to ensure their smooth integration. Structured approaches for the development, application, integration, and maintenance of software products can be found in various software development methodologies (e.g. Cohen et al., 2004; Kniberg, 2011; Rubin, 2012). These approaches strive to improve the development process and quality of complex software by defining the tasks and order of specification, design, implementation, and application in the technical development, and assigning specific roles to developers, users, and coordinators. However, software engineering practices are only slowly finding their way into the environmental modelling domain (e.g. Verweij et al., 2010). Modellers should become more aware of modelling practices (e.g. Refsgaard and Henriksen, 2004; Jakeman et al., 2006; Scholten et al., 2007) as well as software development practices (e.g. Szyperski, 2002; Beydeda et al., 2005) to improve the scientific and technical quality of their models.

6. Discussion and conclusion

We presented an architectural design and prototype implementation of a component-based software framework for the exploratory development of integrated models. The framework provides a unified environment supporting the building and coupling of model components with different temporal and spatial discretisations. The case study example of forest fire demonstrates the flexibility of the framework by straightforward modification and extension of model components.

The framework supports the task of the technical model development as part of the development cycle of integrated models (c.f. Jørgensen and Bendoricchio, 2001; Jakeman et al., 2006; Schmolke et al., 2010). With the framework, a model builder can use the built-in raster-based data types and operations to construct spatio-temporal model components. With component-based design and standardised interfaces, these model components can be straightforward coupled into integrated models. A tedious consolidation of model component packages (e.g. Fall and Fall, 2001; Pullar, 2003; Sklar, 2007; ArcGIS, 2013; ExtendSim, 2013) and coupling frameworks (e.g. Moore and Tindall, 2005; Warner et al., 2008; Hinkel, 2009) can be avoided in this way. The presented concepts of map algebra, model algebra, and accumulators for the scale transfer between components are in general independent from a specific technical implementation. By using a generic high-level scripting environment, we reduce the application barrier for non-software experts compared to the frameworks based on system programming languages as used for example in Collins et al. (2005), Watson and Rahman (2004) or Lindenschmidt et al. (2005). Moreover, high-level languages offer in general faster development times and result in a less error prone development of models.

With Python as development environment, we use a widely applied, platform independent and open-source scripting language. Python provides a large standard library and a wide range of commercial and free extensions for the development of environmental models (e.g. ArcGIS, 2013; SciPy, 2013; Oliphant, 2007; Karssenberg et al., 2010; Van Der Walt et al., 2011; Kraft et al., 2011). The language has a clean syntax making code intuitively comprehensible for non-expert programmers (c.f. Prechelt, 2000; Fangohr, 2004; Loui, 2008; Millman and Aivazis, 2011; Lin, 2012). An advantage compared to compiled languages is the optional use of an interactive Python shell (e.g. Pérez and Granger, 2007). The shell provides an environment for interactive exploration with direct feedback during model development. Interactive shells provide flexibility for exploratory modelling without the traditional cycle of implementation, compilation and execution required by conventional system programming languages (Pérez and Granger, 2007). Modellers can more easily modify and extend their models and are therefore able to respond to changing requirements such as modifications to the conceptual model. Modern software engineering practices such as agile development processes (e.g. Killcoyne and Boyle, 2009; De Kok et al., 2010; Verweij et al., 2010; Scheller et al., 2010) can thus be adopted by environmental scientists without advanced software engineering skills.

Table 7
Implementation of the `Logging` component. The initialisation of the model component is omitted.

```
10  def runTimestep(self):
11      potentialBiomass = self.availableBioMass > self.bioThreshold
12      potHarvest = self.passableCosts & potentialBiomass
13      toHarvest = ifthenelse(uniform(self.passableCosts)
        < 0.2, potHarvest, 0)
14      self.harvested = ifthenelse(toHarvest, self.availableBioMass, 0)
15      self.report(maptotal(self.harvested), "totalMass")
16      harvCost = ifthen(self.harvested, self.costs)
17      self.report(maptotal(harvCost), "totalCosts")
```

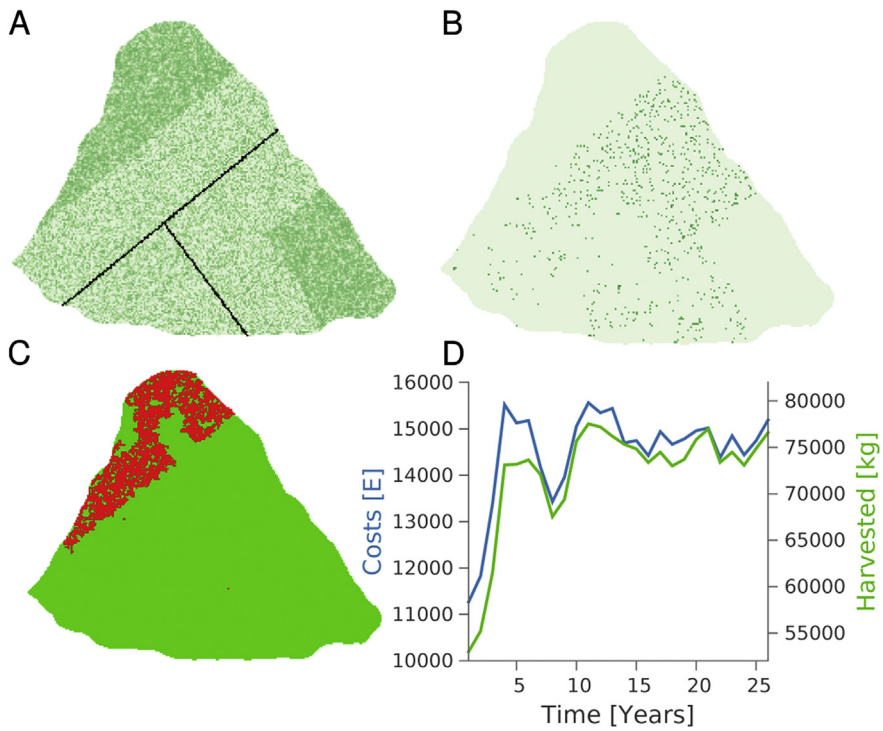

Fig. 11. Outputs of the integrated model extended by the harvest model component. (A) Growth variable from the `Biomass` component with overlayed road infrastructure. (B) Reduced harvesting area in the `Logging` component due to a fire incident. (C) Accumulated output of the `Fire` component. (D) Trend of total costs and harvested biomass.

The features of a generic scripting language as it is used here can also be applied to glue existing software tools into a collaborative environment. Examples of this approach are given by Best et al. (2007), Roberts et al. (2010) or Abdella and Alfredsen (2010), where proprietary and open source packages are combined to provide spatial analysis capabilities. However, such a glueing approach is only feasible when a limited number of components need to be coupled (e.g. Cerco et al., 2010). In our approach, we resolve this problem by extending Python with a framework for the design of model building blocks as well as the execution management required for the development of integrated models. Therefore, the model builder is relieved from the complex task of arranging the scheduling for a larger number of model components with feedback processes. In addition, our more regulated approach with regard to the development of component interfaces is beneficial for the reuse of model components and avoids a repeated implementation per case. Nevertheless, our approach still supports the glueing task of Python as used for example by Roberts et al. (2010). By applying the wrappers, an external, independent executable can be integrated via file system access and a system call (e.g. Hahn et al., 2009; Schmitz et al., 2009). However, substantial work needs to be done by the model builder to transform the variables of the model component interfaces to the input and output files of the external executable.

The scheduler of our framework supports the ordered execution of components and accumulators. The accumulator concept provided by the framework can be used for temporal aggregation when linking different model components. As a consequence, it is no longer necessary to modify process descriptions and to adapt the time steps of components to fit output and input data between coupled components. The separation between the component outputs and accumulators allows for a more generic application and straightforward reuse of existing model components. The case study focuses on the temporal facets of the accumulator, although the accumulator can be utilised as adaptor as well. In this case, the accumulator expresses conversion processes such as spatial resampling of raster variables.

The modelling framework provides accumulators as technical means to resolve spatio-temporal discrepancies. A modeller is able to construct model components operating at their characteristic scales (c.f. Blöschl and Sivapalan, 1995), and to couple these resulting in integrated models. The scientific assessment of feedback effects and the influence of multiple spatial and temporal scales in complex integrated models, however, is not yet fully evolved (Ewert et al., 2009). Modellers need to be aware that a coupling of model components, even if their construction followed good modelling practises, can produce complex patterns (Laniak et al., 1997), and that several problems need to be considered in the evaluation of integrated models. For example, model outcomes can depend on the selection of spatial and temporal discretisations (e.g. Hessel, 2005). The stability of integrated models can be affected if a coupling is not applied over a limited range of scales or in specific situations (c.f. Wainwright and Mulligan, 2004), as discussed for example in rainfall-runoff modelling (Beven, 2004; Chow et al., 1988). Fundamental domain principles such as the conservation of mass in hydrological modelling should not be violated by the coupling of misaligned components, as raised for example by Elag et al. (2011). Further studies are needed to investigate how model composition, the selection of time steps, and the accumulating processes contribute to the sensitivity and uncertainty of integrated models.

The use of model components eases the coupling and reuse of model components within the presented framework. However, the combination of model and framework code for model components does not facilitate their direct integration into other software frameworks (c.f. Lloyd et al., 2011). The interoperability of developed model components with respect to other modelling frameworks is not fully incorporated in the prototype implementation, and technical and semantic barriers still need to be resolved. Nevertheless, it is feasible to couple model components developed

in our framework to other frameworks with limited effort. We briefly discuss potential approaches to enable interaction with components developed in other software environments. First of all, our model classes can be linked to other code within the Python environment. Following object-oriented conventions, coupling of our model components to other Python classes is straightforward by calling methods of the model components. Secondly, the Python model classes can be integrated with frameworks developed in C, C++ or Fortran. For these system programming languages, Python classes can interface for example with the help of automatically generated language bindings. The Python model components can therefore be mapped into frameworks that adopt a component-based development approach like the Integrated Modelling Architecture (IMA; Villa, 2007). Thirdly, the integration of Python model components with an interface standard such as OpenMI (Moore and Tindall, 2005; Gregersen et al., 2007) can be accomplished by mapping the input and output interfaces of our components to OpenMI's equivalents InputExchangeItems and OutputExchangeItems. The Python model components can replicate the ILinkableComponent interface of OpenMI and thus build compliant building blocks, an approach similar to the one presented by Castronova and Goodall (2010). In general, the OpenMI standard is programming language independent and C# and Java implementations are available. However, to integrate Python components, substantial effort by creating a Python reference implementation of the OpenMI standard or by coupling Python and C# within the .Net environment is required. Interfacing OpenMI compliant components in different runtime environments such as Java and Python is also not straightforward. It is thus desirable to bridge the technical barrier of different runtime environments to allow for a coupling of model components developed in virtually any programming language. A fourth approach is therefore to retain defined component interfaces and to realise the information exchange between model components by programming language independent network protocols. This can be achieved by incorporating libraries such as the TDT (Hinkel, 2009) or the ICE (Henning, 2004) into integrated modelling frameworks.

The above-mentioned options are technically demanding, while a domain specialist as model builder would profit from a more conceptual description of model components and ideally their incorporated processes. A step in this direction is to embed formalised descriptions of model components and their interactions with the help of ontologies (e.g. Madin et al., 2008; Buccella et al., 2009; Janssen et al., 2009). While ontologies are mainly used to describe environmental information such as observed ecological data (e.g. Madin et al., 2007) or to formalise spatial and temporal properties (e.g. Miralles et al., 2010), ontologies also can be a means to bridge different modelling frameworks (Rizzoli et al., 2008) or ease the implementation of multi-paradigm models (Villa et al., 2009) as in integrating object-based and field-based models (e.g. Kjenstad, 2006). Formalised descriptions are integrated into several frameworks. The IMA (Villa, 2007), for example, requires interface compliance at the programming level, and enhancement of model components by semantic standards such as the Web Ontology Language (OWL, 2004). However, ontologies for integrated modelling frameworks supporting flexible construction and broad assessment required for example in uncertainty estimation are not fully evolved, although individual approaches are available (e.g. Williams et al., 2009; Gruninger and Tan, 2009; Athanasiadis et al., 2011). Future research will therefore focus on extending the presented framework with scheduling schemes and accompanying ontologies that allow execution and analysis of integrated models with techniques such as Monte Carlo analysis and data assimilation (e.g. Doucet et al., 2001; Moradkhani et al., 2005; Karssenberg et al., 2010).

Acknowledgements

This research is financially supported by the Flemish Institute for Technological Research (VITO). We thank Guy Engelen (VITO) and Johannes van der Kwast (VITO; now at UNESCO-IHE Institute for Water Education) for useful discussions and their assistance, and Noemí Lana-Renault (Universidad de La Rioja) for the provision of the catchment data set. We also thank the reviewers for their constructive comments to the manuscript.

References

Abdella, Y., Alfredsen, K., 2010. A GIS toolset for automated processing and analysis of radar precipitation data. Computers & Geosciences 36 (4), 422–429.
ArcGIS, 2013. Environmental Systems Research Institute. URL. http://www.esri.com/.
Argent, R.M., 2004. An overview of model integration for environmental applications – components, frameworks and semantics. Environmental Modelling & Software 19 (3), 219–234.
Argent, R.M., 2005. A case study of environmental modelling and simulation using transplantable components. Environmental Modelling & Software 20 (12), 1514–1523.
Argent, R.M., Perraud, J.-M., Rahman, J.M., Grayson, R.B., Podger, G.M., 2009. A new approach to water quality modelling and environmental decision support systems. Environmental Modelling & Software 24 (7), 809–818.
Athanasiadis, I.N., Rizzoli, A.-E., Donatelli, M., Carlini, L., 2011. Enriching environmental software model interfaces through ontology-based tools. International Journal of Applied Systemic Studies 4 (1–2), 94–105.
Bach, M.J., 1986. The Design of the Unix Operating System. Prentice-Hall.
Bäcker, A., 2007. Computational physics education with python. Computing in Science and Engineering 9 (3), 30–33.
Beck, M.B., Jakeman, A.J., McAleer, M.J., 1993. Construction and evaluation of models of environmental systems. In: Beck, M.B., Jakeman, A.J., McAleer, M.J. (Eds.), Modelling Change in Environmental Systems. John Wiley & Sons Ltd., New York, pp. 3–35.
Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S., Smith, K., 2011. Cython: the best of both worlds. Computing in Science and Engineering 13 (2), 31–39.
Best, B.D., Halpin, P.N., Fujioka, E., Read, A.J., Qian, S.S., Hazen, L.J., Schick, R.S., 2007. Geospatial web services within a scientific workflow: predicting marine mammal habitats in a dynamic environment. Ecological Informatics 2 (3), 210–223.
Beven, K.J., 2004. Rainfall-runoff Modelling: the Primer. Wiley.
Beydeda, S., Book, M., Gruhn, V. (Eds.), 2005. Model-driven Software Development. Springer.
Bierkens, M.F.P., Finke, P.A., de Willigen, P., 2000. Upscaling and Downscaling Methods for Environmental Research. Kluwer.
Blöschl, G., Sivapalan, M., 1995. Scale issues in hydrological modelling: a review. Hydrological Processes 9 (3–4), 251–290.
Bolte, J.P., Hulse, D.W., Gregory, S.V., Smith, C., 2007. Modeling biocomplexity – actors, landscapes and alternative futures. Environmental Modelling & Software 22 (5), 570–579.
Booch, G., Rumbaugh, J., Jacobson, I., 2005. Unified Modeling Language User Guide, second ed. Addison-Wesley.
Boost.Python, 2012. Boost C++ Libraries. URL. http://www.boost.org/libs/python/.
Brooke, A., Kendrick, D., Meeraus, A., Raman, R., 1998. GAMS, a User's Guide. GAMS Development Corporation.
Buccella, A., Cechich, A., Fillottrani, P., 2009. Ontology-driven geographic information integration: a survey of current approaches. Computers & Geosciences 35 (4), 710–723.
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.
Burrough, P.A., McDonnell, R.A., 1998. Principles of Geographical Information Systems, second ed. Oxford University Press.
Carrera-Hernández, J.J., Gaskin, S.J., 2006. The groundwater modeling tool for GRASS (GMTG): open source groundwater flow modeling. Computers & Geosciences 32 (3), 339–351.
Castronova, A.M., Goodall, J.L., 2010. A generic approach for developing process-level hydrologic modeling components. Environmental Modelling & Software 25 (7), 819–825.
Cerco, C.F., Tillman, D., Hagy, J.D., 2010. Coupling and comparing a spatially- and temporally-detailed eutrophication model with an ecosystem network model: an initial application to Chesapeake Bay. Environmental Modelling & Software 25 (4), 562–572.
Cerveira Cordeiro, J., Câmara, G., Moura de Freitas, U., Almeida, F., 2009. Yet another map algebra. GeoInformatica 13 (2), 183–202.
Chow, V.T., Maidment, D.R., Mays, L.W., 1988. Applied Hydrology. McGraw-Hill, New York.
Cohen, D., Lindvall, M., Costa, P., 2004. An introduction to agile methods. Advances in Computers 62 (C), 1–66.
Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., da Silva, A., 2005. Design and implementation of components in the

Earth System Modeling Framework. International Journal of High Performance Computing Applications 19 (3), 341–350.
Dakos, V., van Nes, E.H., Donangelo, R., Fort, H., Scheffer, M., 2009. Spatial correlation as leading indicator of catastrophic shifts. Theoretical Ecology 3 (3), 163–174.
De Kok, J.-L., Engelen, G., Maes, J., 2010. Towards model component reuse for the design of simulation models – a case study for ICZM. In: Swayne, D.A., Yang, W., Voinov, A.A., Rizzoli, A., Filatova, T. (Eds.), Proceedings of the iEMSs Fourth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2010). International Environmental Modelling and Software Society, Ottawa, Canada, pp. 1215–1222.
Doucet, A., de Freitas, N., Gordon, N., 2001. Sequential Monte Carlo Methods in Practice. In: 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 Resources Research 47, W12520.
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. Environmental Science & Policy 12 (5), 546–561.
ExtendSim, 2013. Imagine that Product Website. URL. http://www.extendsim.com/.
Fall, A., Fall, J., 2001. A domain-specific language for models of landscape dynamics. Ecological Modelling 141 (1–3), 1–18.
Fangohr, H., 2004. A comparison of C, MATLAB, and python as teaching languages in engineering. In: Bubak, M., van Albada, G., Sloot, P., Dongarra, J. (Eds.), Computational Science – ICCS 2004. Vol. 3039 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp. 1210–1217.
Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-oriented Software. Addison Wesley.
GDAL Development Team, 2013. GDAL – Geospatial Data Abstraction Library. URL. http://www.gdal.org/.
Gleeson, T., Wada, Y., Bierkens, M.F.P., van Beek, L.P.H., 2012. Water balance of global aquifers revealed by groundwater footprint. Nature 488 (7410), 197–200.
Gregersen, J.B., Gijsbers, P.J.A., Westen, S.J.P., 2007. OpenMI: open modelling interface. Journal of Hydroinformatics 9 (3), 175–191.
Gruninger, M., Tan, X., 2009. Reasoning about partially ordered web service activities in PSL. Lecture Notes in Computer Science 5926, 231–245.
Hahn, B.M., Kofalk, S., Kok, J.-L.D., 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, pp. 113–136.
HDF5, 2010. Hierarchical Data Format Version 5. The HDF Group, pp. 2000–2010. URL. http://www.hdfgroup.org/HDF5.
Henning, M., 2004. A new approach to object-oriented middleware. IEEE Internet Computing 8 (1), 66–75.
Hessel, R., 2005. Effects of grid cell size and time step length on simulation results of the Limburg soil erosion model (LISEM). Hydrological Processes 19 (15), 3037–3049.
Hiemstra, P.H., Karssenberg, D., van Dijk, A., 2011. Assimilation of observations of radiation level into an atmospheric transport model: a case study with the particle filter and the ETEX tracer dataset. Atmospheric Environment 45 (34), 6149–6157.
Hinkel, J., 2009. The PIAM approach to modular integrated assessment modelling. Environmental Modelling & Software 24 (6), 739–748.
Hunter, J.D., 2007. Matplotlib: a 2D graphics environment. Computing in Science and Engineering 9 (3), 99–104.
Jakeman, A.J., Letcher, R.A., Norton, J.P., 2006. Ten iterative steps in development and evaluation of environmental models. Environmental Modelling & Software 21 (5), 602–614.
Janssen, S., Ewert, F., Li, H., Athanasiadis, I.N., Wien, J.J.F., Thérond, O., Knapen, M.J.R., Bezlepkina, I., Alkan-Olsson, J., Rizzoli, A.E., Belhouchette, H., Svensson, M., van Ittersum, M.K., 2009. Defining assessment projects and scenarios for policy support: use of ontology in integrated assessment and modelling. Environmental Modelling & Software 24 (12), 1491–1500.
Jørgensen, S.E., Bendoricchio, G., 2001. Fundamentals of Ecological Modelling. Elsevier, Amsterdam.
Karafyllidis, I., Thanailakis, A., 1997. A model for predicting forest fire spreading using cellular automata. Ecological Modelling 99 (1), 87–97.
Karssenberg, D., 2002. The value of environmental modelling languages for building distributed hydrological models. Hydrological Processes 16 (14), 2751–2766.
Karssenberg, D., de Jong, K., 2005. Dynamic environmental modelling in GIS: 1. Modelling in three spatial dimensions. International Journal of Geographical Information Science 19 (5), 559–579.
Karssenberg, D., de Jong, K., van der Kwast, J., 2007. Modelling landscape dynamics with Python. International Journal of Geographical Information Science 21 (5), 483–495.
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. Environmental Modelling & Software 25 (4), 489–502.
Killcoyne, S., Boyle, J., 2009. Managing chaos: lessons learned developing software in the life sciences. Computing in Science and Engineering 11 (6), 20–29.
Kjenstad, K., 2006. On the integration of object-based models and field-based models in GIS. International Journal of Geographical Information Science 20 (5), 491–509.
Kniberg, H., 2011. Lean from the Trenches: Managing Large-scale Projects with Kanban. Pragmatic Bookshelf.
Koomsap, P., Shaikh, N.I., Prabhu, V.V., 2005. Integrated process control and condition-based maintenance scheduler for distributed manufacturing control systems. International Journal of Production Research 43 (8), 1625–1641.
Kraft, P., Vaché, K.B., Frede, H.-G., Breuer, L., 2011. CMF: a hydrological programming language extension for integrated catchment models. Environmental Modelling & Software 26 (6), 828–830.
Langtangen, H.P., 2007. Python Scripting for Computational Science, third ed. Springer.
Laniak, G.F., Droppo, J.G., Faillace, E.R., Gnanapragasam, E.K., Mills, W.B., Strenge, D.L., Whelan, G., Yu, C., 1997. An overview of a multimedia benchmarking analysis for three risk assessment models: RESRAD, MMSOILS, and MEPAS. Risk Analysis 17 (2), 203–214.
Laniak, G.F., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P., Whelan, G., Geller, G., Quinn, N., Blind, M., Peckham, S., Reaney, S., Gaber, N., Kennedy, R., Hughes, A., 2013. Integrated environmental modeling: a vision and roadmap for the future. Environmental Modelling & Software 39, 3–23.
Lin, J.W.-B., 2009. qtcm 0.1.2: a python implementation of the Neelin-Zeng Quasi-Equilibrium Tropical Circulation model. Geoscientific Model Development 2 (1), 1–11.
Lin, J.W.-B., 2012. Why python is the next wave in earth sciences computing. Bulletin of the American Meteorological Society 93 (12), 1823–1824.
Lindenschmidt, K.-E., Rauberg, J., Hesser, F.B., 2005. Extending uncertainty analysis of a hydrodynamic-water quality modelling system using high level Architecture (HLA). Water Quality Research Journal of Canada 40 (1), 59–70.
Lloyd, W., David, O., Ascough II, J.C., Rojas, K.W., Carlson, J.R., Leavesley, G.H., Krause, P., Green, T.R., Ahuja, L.R., 2011. Environmental modeling framework invasiveness: analysis and implications. Environmental Modelling & Software 26 (10), 1240–1250.
Loui, R.P., 2008. In praise of scripting: real programming pragmatism. Computer 41, 22–26.
Madin, J., Bowers, S., Schildhauer, M., Krivov, S., Pennington, D., Villa, F., 2007. An ontology for describing and synthesizing ecological observation data. Ecological Informatics 2 (3), 279–296.
Madin, J.S., Bowers, S., Schildhauer, M.P., Jones, M.B., 2008. Advancing ecological research with ontologies. Trends in Ecology & Evolution 23 (3), 159–168.
May, R.M., 1977. Thresholds and breakpoints in ecosystems with a multiplicity of stable states. Nature 269 (5628), 471–477.
Mennis, J., 2010. Multidimensional map algebra: design and implementation of a spatio-temporal GIS processing language. Transactions in GIS 14 (1), 1–21.
Millman, K.J., Aivazis, M., 2011. Python for scientists and engineers. Computing in Science and Engineering 13 (2), 9–12.
Miralles, A., Pinet, F., Bédard, Y., 2010. Describing spatio-temporal phenomena for environmental system development: an overview of today's needs and solutions. International Journal of Agricultural and Environmental Information Systems 1 (2), 68–84.
Moore, R.V., Tindall, C.I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environmental Science & Policy 8 (3), 279–286.
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 Resources Research 41, W05012.
North, M.J., Collier, N.T., Vos, J.R., 2006. Experiences creating three implementations of the repast agent modeling toolkit. ACM Transactions on Modeling and Computer Simulation 16 (1), 1–25.
OGR, 2013. Simple Feature Library. URL. http://www.gdal.org/ogr/index.html.
Oliphant, T.E., 2007. Python for scientific computing. Computing in Science and Engineering 9 (3), 10–20.
OWL, 2004. OWL Web Ontology Language Guide. URL. http://www.w3.org/TR/owl-guide/.
Papajorgji, P., 2005. A plug and play approach for developing environmental models. Environmental Modelling & Software 20 (10), 1353–1357.
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. International Journal of Geographical Information Science 21 (5), 515–527.
Pérez, F., Granger, B.E., 2007. IPython: a system for interactive scientific computing. Computing in Science and Engineering 9 (3), 21–29.
Pérez, F., Granger, B.E., Hunter, J.D., 2011. Python: an ecosystem for scientific computing. Computing in Science and Engineering 13 (2), 13–21.
Peterson, P., 2009. F2PY: a tool for connecting Fortran and Python programs. International Journal of Computational Science and Engineering 4 (4), 296–305.
Prechelt, L., 2000. Empirical comparison of seven programming languages. Computer 33 (10), 23–29.
Pullar, D., 2001. MapScript: a map algebra programming language incorporating neighborhood analysis. GeoInformatica 5 (2), 145–163.
Pullar, D., 2003. Simulation modelling applied to runoff modelling using MapScript. Transactions in GIS 7 (2), 267–283.
Python, 2013. Python Programming Language Website. URL. http://www.python.org/.
Quantum GIS Development Team, 2013. Quantum GIS Geographic Information System. Open Source Geospatial Foundation. URL. http://qgis.osgeo.org.
Refsgaard, J.C., Henriksen, H.J., 2004. Modelling guidelines – terminology and guiding principles. Advances in Water Resources 27 (1), 71–82.

Rizzoli, A.E., Donatelli, M., Athanasiadis, I.N., Villa, F., Huber, D., 2008. Semantic links in integrated modelling frameworks. Mathematics and Computers in Simulation 78 (2–3), 412–423.
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++. Environmental Modelling & Software 25 (10), 1197–1207.
RPy, 2011. Python Interface to the R Programming Language. URL. http://rpy.sourceforge.net/.
Rubin, K.S., 2012. Essential Scrum: a Practical Guide to the Most Popular Agile Process. Addison-Wesley.
Saiful Islam, A.K.M., Piasecki, M., 2006. A generic metadata description for hydrodynamic model data. Journal of Hydroinformatics 8 (2), 141–148.
Scheller, R.M., Sturtevant, B.R., Gustafson, E.J., Ward, B.C., Mladenoff, D.J., 2010. Increasing the reliability of ecological models using modern software engineering techniques. Frontiers in Ecology and the Environment 8 (5), 253–260.
Schmitz, O., Karssenberg, D., de Kok, J.-L., 2012. Towards integrated model building with semantically annotated components. In: Seppelt, R., Voinov, A.A., Lange, S., Bankamp, D. (Eds.), International Congress on Environmental Modelling and Software. Managing Resources of a Limited Planet: Pathways and Visions Under Uncertainty, Sixth Biennial Meeting. International Environmental Modelling and Software Society (iEMSs), Leipzig, Germany, pp. 2403–2412.
Schmitz, O., Karssenberg, D., van Deursen, W.P.A., Wesseling, C.G., 2009. Linking external components to a spatio-temporal modelling framework: coupling MODFLOW and PCRaster. Environmental Modelling & Software 24 (9), 1088–1099.
Schmolke, A., Thorbek, P., DeAngelis, D.L., Grimm, V., 2010. Ecological models supporting environmental decision making: a strategy for the future. Trends in Ecology & Evolution 25 (8), 479–486.
Scholten, H., Kassahun, A., Refsgaard, J.C., Kargas, T., Gavardinas, C., Beulens, A.J., 2007. A methodology to support multidisciplinary model-based water management. Environmental Modelling & Software 22 (5), 743–759.
Schroeder, W.J., Avila, L.S., Hoffman, W., 2000. Visualizing with VTK: a tutorial. IEEE Computer Graphics and Applications 20 (5), 20–27.
SciPy, 2013. Scientific Tools for Python. URL. http://www.scipy.org/.
Sklar, E., 2007. Software review: NetLogo, a multi-agent simulation environment. Artificial Life 13 (3), 303–311.
Szyperski, C., 2002. Component Software: Beyond Object-oriented Programming, second ed. Addison Wesley.
Tanenbaum, A.S., 1992. Modern Operating Systems. Prentice-Hall.
Tomlin, C.D., 1990. Geographic Information Systems and Cartographic Modeling. Prentice Hall.
Torrey, L.A., Coleman, J., Miller, B.P., 2007. A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. Software-practice and Experience 37 (4), 347–364.
Uschold, M., Gruninger, M., 1996. Ontologies: principles, methods and applications. Knowledge Engineering Review 11 (2), 93–136.
van Beek, L.P.H., Wada, Y., Bierkens, M.F.P., 2011. Global monthly water stress: 1. Water balance and water availability. Water Resources Research 47 (7), W07517.
van Delden, H., Luja, P., Engelen, G., 2007. Integration of multi-scale dynamic spatial models of socio-economic and physical processes for river basin management. Environmental Modelling & Software 22 (2), 223–238.
van der Knijff, J.M., Younis, J., de Roo, A.P.J., 2010. LISFLOOD: a GIS-based distributed model for river basin scale water balance and flood simulation. International Journal of Geographical Information Science 24 (2), 189–212.
Van Der Walt, S., Colbert, S.C., Varoquaux, G., 2011. The NumPy array: a structure for efficient numerical computation. Computing in Science and Engineering 13 (2), 22–30.
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. Computers, Environment and Urban Systems 36 (1), 30–46.
Verweij, P.J.F.M., Knapen, M.J.R., de Winter, W.P., Wien, J.J.F., te Roller, J.A., Sieber, S., Jansen, J.M.L., 2010. An IT perspective on integrated environmental modelling: the SIAT case. Ecological Modelling 221 (18), 2167–2176.
Villa, F., 2007. A semantic framework and software design to enable the transparent integration, reorganization and discovery of natural systems knowledge. Journal of Intelligent Information Systems 29 (1), 79–96.
Villa, F., Athanasiadis, I.N., Rizzoli, A.E., 2009. Modelling with knowledge: a review of emerging semantic approaches to environmental modelling. Environmental Modelling & Software 24 (5), 577–587.
Voinov, A., Fitz, C., Boumans, R., Costanza, R., 2004. Modular ecosystem modeling. Environmental Modelling & Software 19 (3), 285–304.
Wainwright, J., Mulligan, M., 2004. Environmental Modelling. Wiley, Chichester.
Warner, J.C., Perlin, N., Skyllingstad, E.D., 2008. Using the Model Coupling Toolkit to couple earth system models. Environmental Modelling & Software 23 (10–11), 1240–1249.
Watson, F.G.R., Rahman, J.M., 2004. Tarsier: a practical software framework for model development, testing and deployment. Environmental Modelling & Software 19 (3), 245–260.
Wesseling, C.G., Karssenberg, D., Burrough, P.A., van Deursen, W.P.A., 1996. Integrating dynamic environmental models in GIS: the development of a dynamic modelling language. Transactions in GIS 1 (1), 40–48.
Williams, M., Cornford, D., Bastin, L., Pebesma, E.J., 2009. Uncertainty Markup Language (UncertML). OGC Discussion Paper, Document Number: 08–122r1.
XML, 2008. Extensible Markup Language (XML) 1.0. W3C. URL. http://www.w3.org/TR/REC-xml/.
Zhao, G., Hörmann, G., Fohrer, N., Gao, J., Li, H., Tian, P., 2011. Application of a simple raster-based hydrological model for streamflow prediction in a humid catchment with polder systems. Water Resources Management 25 (2), 661–676.