Environmental Modelling & Software 24 (2009) 1088–1099

Contents lists available at ScienceDirect

Environmental Modelling & Software

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

Linking external components to a spatio-temporal modelling framework: Coupling MODFLOW and PCRaster

O. Schmitz [a,*], D. Karssenberg [a], W.P.A. van Deursen [b], C.G. Wesseling [b]

[a] Department of Physical Geography, Faculty of Geosciences, Utrecht University, Heidelberglaan 2, PO Box 80115, 3508 TC Utrecht, The Netherlands
[b] PCRaster Environmental Software Consultancy, Van Swindenstraat 97, 3514 XP Utrecht, The Netherlands

ARTICLE INFO

Article history:
Received 16 July 2008
Received in revised form
12 February 2009
Accepted 27 February 2009
Available online 11 April 2009

Keywords:
Environmental modelling
PCRaster
Python
MODFLOW
Integrated modelling

ABSTRACT

An important step in the procedure of building an environmental model is the transformation of a conceptual model into a numerical simulation. To simplify model construction a framework is required that relieves the model developer from software engineering concerns. In addition, as the demand for a holistic understanding of environmental systems increases, access to external model components is necessary in order to construct integrated models.

We present a modelling framework that provides two- and three-dimensional building blocks for construction of spatio-temporal models. Two different modelling languages available in the framework, the first tailored and the second an enhanced Python scripting language, allow the development and modification of models. We explain for both languages the interfaces allowing to link specialised model components and thus extending the functionality of the framework. We demonstrate the coupling of external components in order to create multicomponent models by the development of the link to the groundwater model MODFLOW and provide results of an integrated catchment model. The approach described is appropriate for constructing integrated models that include a coupling of a small number of components.

Software availability

Name: PCRaster
Developer: Department of Physical Geography, PO Box 80115, 3508 TC Utrecht, The Netherlands; PCRaster Environmental Software Consultancy, Van Swindenstraat 97, 3514 XP Utrecht, The Netherlands
Contact: o.schmitz@geo.uu.nl (MODFLOW extension), d.karssenberg@geo.uu.nl
Required software:
PCRaster: Windows (free), Linux and UNIX on request; http://pcraster.geo.uu.nl/
Python: all major platforms; http://www.python.org/
Online courses at http://pcraster.geo.uu.nl/courses/courses.html
Example scripts and data sets are included in the install package

1. Introduction

Numerical spatio-temporal models simulating processes in the geographic domain are important tools to improve our understanding of geographic systems. They are used in disciplines as diverse as human and physical geography, ecology, environmental sciences and hydrology. Each discipline uses models representative for components of their specific geographic field. For instance, human geographers model the socio-economic system to represent urban sprawl, ecologists model the evolution of vegetation under environmental stress, and hydrologists use simulations of water flow in catchments. However, the need to link models of individual components is becoming more and more urgent. This is driven by the necessity to understand interactions in large geographic systems, particularly in environmental management and planning. Such large systems can only be modelled by explicitly representing the interrelations between different, smaller, component models. This is referred to as integrated modelling (Argent, 2004).

Component models are often developed to describe particular environmental processes and are used by a limited user group. Integrated models representing a number of interacting processes optionally include social, economic, infrastructure and governance aspects (McIntosh et al., 2007). The increase of involved domains equally increases the number of involved user groups and their

* Corresponding author. Tel.: +31 30 253 9363; fax: +31 30 253 1145.
E-mail address: o.schmitz@geo.uu.nl (O. Schmitz).

1364-8152/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.envsoft.2009.02.018

specific perceptions of the integrated model. An example for divergent perceptions of integrated models is decision support systems where the policy view is originated in the value-driven perceptual domain while the research view is originated in the theoretical domain (Oxley et al., 2004). The problems related to the transfer of scientific knowledge into policy models (e.g. McIntosh et al., 2005) are not addressed here. In this paper we discuss the development of integrated research models by scientists specialised in their domain, for instance hydrology or ecology.

In general, both component models and integrated models need to provide a spatial representation in two or three dimensions, and need to simulate changes over time. The representation of the time domain is referred to as transient or dynamic modelling (Beck et al., 1993; Karssenberg and de Jong, 2005; Burrough, 1998). In dynamic modelling, which is the term used here, the state of a model is iteratively updated over a set of discretised timesteps by equations representing real world processes such as urban sprawl, vegetation growth, or groundwater flow. Dynamic models either use continuous fields or discrete entities to represent the geographic domain. Discrete entities are widely applied in agent based and individual based modelling (c.f. Benenson and Torrens, 2004; Grimm and Railsback, 2005). The focus in this work is on models that use continuous fields discretised in raster cells utilising as update rules for instance cellular automata, differential equations or rule based spatial functions.

The aim of model building is to find the optimal set of equations to represent real world processes, given the purpose of the model and the data available. This is often done in an explorative way, whereby different sets of equations are evaluated and the set of equations is selected that gives the best performance of the model in terms of its predictive capability, calibration and validation results, runtime, and input data required (Wainwright and Mulligan, 2004). Thus, in model building it is important to have a tool that is flexible such that a wide range of different model equations can be programmed, (re-)combined and tested in relatively short time (Karssenberg, 2002; Argent, 2004). Such a flexible tool is also required when adjustments to existing models need to be made, for instance when new data become available that need to be fed to the model, or when new scientific understanding can be used to improve the representation of real world processes in the model.

Although in principle any programming language could be used as model construction tool, modelling frameworks (Argent, 2004; Rizzoli et al., 2008) have been developed for the purpose of model construction (e.g. MATLAB, 2005; Wang and Pullar, 2005; PCRaster, 2008; Leavesley et al., 1998; Repast, 2008; Hurkens et al., 2008). For model construction, three features of a modelling framework are essential (Argent, 2004; Karssenberg, 2002): native operations on spatial entities, a modelling language for combining these operations in order to build dynamic models, and a means to link with external programs or models.

The range of native operations included can vary from low-level approaches where the operations are mathematical operators on arrays (e.g. MATLAB, 2005) to high level approaches that provide operations representing spatial processes on 2D and 3D entities representing a landscape (Pullar, 2004; PCRaster, 2008; ArcGIS, 2008). The use of high level operations minimises the time required to program models, but comes with the risk that not all types of models can be implemented because the number of different operations is too small, or the operations do not provide enough flexibility to support application in a wide range of application domains (c.f. Karssenberg, 2002).

The language provided to construct the models using the native operations is also an important factor determining the efficiency of model construction and the possible user group of the modelling framework. Low-level system programming languages like C, Fortran or C++ require that many technical details, not related to environmental modelling, which need to be defined in the program. As a result, combining the operations requires a strong background in programming. On the other hand, frameworks based on higher level languages such as Geographic Information System (GIS) languages (ArcGIS, 2008; GRASS, 2008) are completely tailored to model construction and include standard functionality for data handling and detailed type checking and in some cases also for iterations over time (e.g. De Vasconcelos et al., 2002; PCRaster, 2008). In this case, the model script resembles very much a technical documentation of the model: it can be written and read without much knowledge of informatics. In general, it is preferable to hide as many technical details from the modeller, so higher level languages are preferable to lower level languages (Karssenberg, 2002).

By combining the operations provided by the modelling framework, the user can construct a large range of models. However, in some cases a model component is required that cannot be constructed from the operations provided by the modelling framework. This model component can be small, for instance a single spatial operation that is not provided by the framework, or it can be large, i.e. a complete spatio-temporal sub-model that needs to interact with the model that is constructed. The latter is often the case with integrated modelling, where a number of models need to communicate by exchanging their model states.

Two approaches can be followed to create a link with external model components. In the first approach, the modelling framework keeps its central role in handling input and output of data and providing control flow, in particular timesteps. The external model component is linked to the framework and is called from within the framework itself. In the second approach, an external modelling interface is used to create the link with an external component. The model constructed in the modelling framework looses its central role and it becomes one of the component models in the modelling interface, just like the external model component to which it needs to be linked. This approach to model integration is possible with the Open Modelling Interface (OpenMI, 2008; Gregersen et al., 2007).

In this paper we describe and evaluate the PCRaster modelling framework (van Deursen, 1995; Karssenberg and de Jong, 2005; PCRaster, 2008). This framework provides an extensive set of operations on two- and three-dimensional continuous fields, and seamlessly integrates the temporal dimension in the operations and the database. For the construction of models from these native operations and, in effect also from external modules, PCRaster comes in two flavours: the PCRcalc modelling language, fully tailored to construction of dynamic spatial models and an integration of the Python scripting language (2008). The PCRcalc language is an example of a dedicated high level language as described above that is fully targeted at model builders by hiding all technical details. The Python language is at a somewhat lower level because it is a generic language. For both flavours of the PCRaster framework we describe features to extend the languages with links to external model components. Here, the modelling framework keeps it central role as described above in the first approach. The procedure to link external components is illustrated with an example hydrological model. The purpose of the example is to create a link with MODFLOW, the widely used software for groundwater flow modelling (McDonald and Harbaugh, 1984).

The aim of this paper is 1) to explain the concepts of dynamic modelling and how operations have been designed following these concepts, 2) to describe the technical implementation in the PCRaster modelling framework (for both PCRcalc and PCRaster Python) that incorporate these operations, 3) to explain how external models can be linked to the modelling framework with the example of MODFLOW, and 4) to evaluate the modelling framework with respect to the usability for model construction whereby we

discuss the use of OpenMI as an alternative strategy to link external components.

2. Concepts of dynamic spatial modelling

To simulate dynamic behaviour of a geographic system a discretisation of a time period into a sequence of timesteps and an iteration over these timesteps are needed. This can be described by executing a functional f at each timestep t_i:

$$Z_{1,\ldots,m}(t_i) = f(Z_{1,\ldots,m}(t_{i-1}), I_{1,\ldots,n}(t_i), P_{1,\ldots,l}) \,\forall\, t_i \quad (1)$$

The model variables $Z_{1,\ldots,m}$ at the new timestep are the result of the state change functional f which can either be an update rule, a differential equation derivative describing the variables as a continuous function or a probabilistic ruleset (Karssenberg and de Jong, 2005). Arguments to that function are a set of parameters $P_{1,\ldots,l}$ and input parameters $I_{1,\ldots,n}$ of the current timestep. Furthermore the variable states of the previous timestep $Z_{1,\ldots,m}(t_{i-1})$ are included in order to model feedback dependencies.

The representation of space basically demands the same properties provided by current GISs and requires native data types to manage two-dimensional map data and three-dimensional volume data. To represent f, a set of native operations is essential (Tomlin, 1990; Wesseling et al., 1996; Pullar, 2003). A consecutive application of the function f updates the model state for each timestep. Spatial and temporal operations on native data types form the building blocks which enable the model developer to construct dynamic models. As an additional requirement visualisation tools that are able to display spatio-temporal data need to be provided. Changes over time can be shown as timeseries graph data or as an animated sequence of maps.

In the following sections we present the PCRaster modelling framework offering two- and three-dimensional data structures and native operations which can be combined to construct f (Equation (1)).

3. PCRaster modelling framework

PCRaster (2008) is a framework for the development of dynamic models. The integrated database organises the storage and management of raster-based two-dimensional maps and three-dimensional block structures. Building on that, the modelling framework offers a large set of native operations that operate on spatial entities and the temporal domain. The operations can be classified into the following groups (Karssenberg and de Jong, 2005; Tomlin, 1990) (see Fig. A1):

(1) Point or local functions that operate on single cells or voxels. Examples are arithmetic functions or all functions returning attributes of a cell as a function of values at the cell itself, for instance the assignment of hydraulic conductivity derived from the lithology of a cell.
(2) Direct neighbourhood or focal functions operating on cells or voxels in a spatially bound neighbourhood. Examples are two- or three-dimensional filters calculating the new cell value as a function of attribute values in the window.
(3) Entire neighbourhood functions. These functions derive attribute values of a cell from attribute values in potentially all cells on a map. Examples are functions solving equations representing flow in the whole modelling domain, or functions calculating distances between cells or voxels.
(4) Functions with a neighbourhood defined by a given topology. These functions calculate cell values from attribute values of cells from a neighbourhood defined by an explicit given topology. Examples are functions transporting material over the local drain direction network representing flow directions over a map (Fig. A3).

In contrast to most GISs, PCRaster provides capabilities for modelling in space and time. Fig. A2 provides a conceptual scheme of a PCRaster model including the model sections, each with a specific role in a model. The initial section holds the code defining the starting values of the model. Initial values can be obtained from disk, or are calculated with spatial operations. The dynamic section holds all operations that are executed in one timestep. Model input can be read from disk, resulting output can be in the form of temporal data or spatio-temporal data. The timer sets and controls the number of iterations over timesteps. After a model run, the results can be displayed as static or dynamic, animated data.

The PCRaster framework is used worldwide for model development at research institutes and academia as well as for education in environmental modelling, including e-learning courses (Karssenberg et al., 2001). Applications of the framework can be found amongst others in hydrological modelling (e.g. LISFLOOD, van der Knijff et al., 2008), geomorphologic modelling (Karssenberg and Bridge, 2008) or wind erosion modelling (Visser et al., 2004). The implementation of various models using PCRaster is discussed in Wesseling et al. (1996), Burrough et al. (2005) or Karssenberg (2002).

3.1. The PCRaster PCRcalc modelling language

3.1.1. Native functions and model script

The PCRcalc modelling language uses the native PCRaster functions in the development environment PCRcalc. The basic layout of a model described in PCRcalc contains five different sections which are shown in the example script of Table A1. The *binding* section links external file names to model parameters and thus allows the exchange of input data without modifications in the remaining sections of the script. The *areamap* section defines the spatial discretisation of a model. In the *timer* section the start time, end time and duration of a model are specified. The *initial* section contains the code for the model setup which means initialising variables by values read from disk, setting constant values or assigning the result of spatial operations. The *dynamic* section finally covers a set of operations describing the system processes to represent f (Equation (1)) for each timestep. Inputs dependent on the timestep are retrieved with the temporal operations, intermediate results can be stored on disk. The initial and dynamic sections contain one or more statements of the type:

```
result Map = native Operation (argument Maps);
```

The combination of several statements allows the formal representation of the processes occurring in a system, as shown in the example of Table A1. Here the initial section is used to create a local drain direction map as shown in Fig. A3 containing the flow direction of each cell in a catchment. The native operation `timeinputscalar` reads the timeseries file `rain.tss` and assigns for each timestep each cell a precipitation value as input data. The native `accuflux` operation calculates the amount of water that flows out of a cell to its neighbour cell in a direction specified in the local drain direction map `ldd`. The modelling framework parses the PCRcalc model script and checks it for syntax errors and optimisation opportunities before it executes the model.

3.1.2. Extending the PCRcalc language with external model components

The Application Programming Interface (API) of PCRaster enables software developers to integrate their models into the PCRaster framework. The API offers a communication layer that

allows to add own model components and enables access and modification of PCRaster data values. The API therefore allows transformation from own data types to that used in PCRaster as well as storage and modification of data independent from PCRaster data types and timesteps. Developers of model components can add functionality to the modelling language by writing libraries in their favourite programming language.

The library is accompanied by an XML manifest file. The XML file is used by the framework to identify the operations the library provides. Table A2 shows an example of a manifest file specifying an extension operation performing a statistical calculation which is not available as native operation. The operation named *colMedian* takes a spatial argument of the data type scalar as input and returns a scalar spatial result. At the beginning of a model script execution the XML manifest file is parsed by the modelling framework. In a second step the type informations of the operations and arguments are tested for correctness.

In addition to the manifest file the developer has to implement the function itself that is callable by the PCRaster framework and organises the data transfer between the modelling framework and the extension library. Table A3 shows a code extract of the library for the operation *colMedian*. The function `pcr_LinkInExecute` is called by the PCRaster framework and provides at method invocation via the array `LinkInTransferArray`, which is holding pointer to result and argument maps, access to the PCRaster data types.

The operation implemented in the library can be used in a PCRcalc script as follows:

```
result:map = extensionName :: colMedian(dem:map);
```

where `extensionName` is the filename of the extension selectable by the library developer.

The API therefore allows the development of more advanced libraries by providing object-orientation with an object state and methods interacting with that state. It is therefore possible to link to complex state persistent model components as will be shown in Section 4.2.

3.2. The PCRaster Python language

3.2.1. The Python language

Unlike PCRcalc, being a tailored modelling language, Python is a generic programming language. Compared to the traditional system programming languages such as Fortran and C, Python liberates the model developer from the need to obtain specialised knowledge about for example the underlying operating system and memory management. Python provides the typical programming language features like loops, control flow and definition of functions as well as object-orientation which allows modularisation, definition of own data types and re-use of specific components. The clean syntax makes Python easily learnable and thus a suitable basis for a modelling language.

3.2.2. Extending Python with model building blocks

To enhance Python with modelling abilities spatial data types and operations must be added. The mixed language programming feature provided by Python allows the integration of C, C++ or Fortran code without the need for modifications of the existing code. We use this approach to provide the spatial operations of the PCRaster framework as a Python extension. The added overhead for the communication between Python and the framework code is of minor importance, as the main runtime of a model is spent on spatial operations and thus in the optimised framework code.

The mixed programming layout requires additional code which organises the communication between the Python language and the PCRaster code. The interface used to expose the objects and methods is the Boost.Python library (2008). Functionality such as operations on each cell of a map is implemented in C++ methods. The Boost.Python framework defines the operation name used in the Python language and specifies the associated C++ class and method. At compile time the library generates code that is callable by the Python C API. Result of the compilation is an extension that can be imported and used in a Python script.

In addition to the algebraic operations on maps a framework for model execution is needed. We provide a set of Python classes that organise the execution of static and dynamic models (c.f. Karssenberg et al., 2007), whereby the structure of the model is similar to the structure of a model written in PCRcalc. Furthermore modules for error propagation and data assimilation are available (Karssenberg et al., 2008a,b).

Table A4 shows the Python version of the PCRcalc runoff model given in Table A1 and demonstrates the combination of Python language features by defining the model with a class and member functions and execution by the dynamic framework.

As well as the map algebra operations of the PCRaster framework are provided as Python extension, external components can be included into model scripts. We again use the Boost.Python library to implement the link to the MODFLOW component as Python extension, as will be shown below.

4. Catchment model case study

In this section we show how the PCRaster framework can be used to construct an integrated catchment model. This model uses hereby a combination of native and extension operations to represent hydrologic processes. Simulation of surface runoff and the calculation of discharge fluxes are done with native operations. As three-dimensional flow equations are not available as native operations within the framework, the groundwater component is simulated by the external application MODFLOW.

4.1. Linking the MODFLOW component

MODFLOW (Harbaugh et al., 2000), developed by the US Geologic Survey (USGS, 2008) and first released in 1984, is a free software package solving three-dimensional groundwater flow equations. Due to its comprehensive documentation, the extensibility and development of additional packages (e.g. Osman and Bruen, 2002; Batelaan and De Smedt, 2004), it is used in a number of scientific research projects (Herzog et al., 2003; Nguyen et al., 2005; Gedeon et al., 2007). Documentation and source code from MODFLOW are available at the website of the USGS (2008).

MODFLOW uses a set of specific packages to define the model. The discretisation package sets the spatial and temporal dimensions. Spatial properties cover the finite difference grid in the horizontal and the number of layers in the vertical orientation. Time is divided into stress periods, during which external stresses like pumping rates are constant, and furthermore into timesteps.

Initial head values and boundary conditions are defined in the basic package. Boundary values define whether the flow in a cell is constant, calculated or not considered. Conductivity and transmissivity values and settings for the rewetting capabilities are specified in the block-centred flow package. Packages simulating stress factors are the well, recharge, river and drain packages. Furthermore solver and control files must be specified for a simulation. Results of a simulation are head and new boundary values for each layer as well as flow terms related to each of the specified packages.

All MODFLOW inputs need to be defined as ASCII input files with a strict format. As it is sometimes cumbersome to create these files, a wide range of commercial and free pre- and postprocessors have

been developed to ease the model creation (e.g. Winston, 1999; Processing Modflow, 2008; VisualMODFLOW, 2008; Carrera-Hernández and Gaskin, 2006). However, these MODFLOW shells do not allow for integration of MODFLOW with other component models that model parts of the system that cannot be represented by MODFLOW, such as the unsaturated zone or surface water.

The link between the PCRaster modelling framework and MODFLOW does allow for integrated modelling, because MODFLOW can be called from within the modelling framework. Groundwater is modelled with MODFLOW, while the native operations of the modelling framework are used to represent the other components of the hydrological system. The MODFLOW stress periods are mapped to the framework timesteps, the modeller is able to choose steady-state or transient simulations of the stress periods. Thus, the modeller is free in choosing a duration of timesteps that matches the available data and process dynamics.

We developed extensions both for the PCRcalc and Python languages using the PCRcalc API and the Python wrapper approaches explained in the Sections 3.1.2 and 3.2.2. The extension updates necessary MODFLOW input files each timestep and reads the output generated by MODFLOW. An executable of MODFLOW 2000 (v1.17) is included in the extension package. Minor modifications to the MODFLOW code were made to suppress status messages to the standard output.

The approach is illustrated below in an example where rainfall and river discharge are represented with the native operations of the modelling framework. This framework is linked to a two layer MODFLOW model to represent groundwater flow.

4.2. PCRaster Modflow

4.2.1. Study area

We used the extension to build an integrated model of the "Utrechtse Heuvelrug" catchment located in the centre of the Netherlands. The model written in the PCRaster PCRcalc modelling language is provided in Table A5 and continued in Table A6. For the PCRaster Python version of the same model we refer to Table A7.

Fig. A4 shows the digital elevation map and the streams in the area. The size of the catchment amounts to 120 km^2. Elevation values range from 2–5 m in the lowland parts to 50 m on top of the ridge. The soils consist mainly of Pleistocene sands and Holocene river deposits. Pasture is the dominating landuse type in the lower parts of the catchment, the upper parts are dominated by dry woodlands. Large parts of the lower area are drained. The surface area sums up to about 5000 cells with a cell length of 150 m. Two layers of variable thickness per cell are used. Five streams, drained areas and one pumping well in the bottom layer are included in the model.

4.2.2. Initial model section

As shown in the model script in Tables A5 and A6, a set of operations arranges the communication with the MODFLOW extension. Settings which are not changing over time like the grid specification and constant values are set in the initial section of a model. Variations in the stress packages are set in the dynamic section.

Specifying the grid dimensions of the discretisation package must be the first activity after `initialise` presets the internal data structures of the extension. The `createBottomLayer` operation defines the bottom layer and takes two maps as arguments containing the bottom and top of the bottom layer elevation values. An additional layer is added with the `addLayer` operation with a top of layer elevation value map as argument. Both operations calculate the thickness values per layer and hence build up the vertical grid dimensions. The horizontal properties like "cell width along rows" are derived from the information about the spatial discretisation integrated in the provided `clone.map`.

The input for the block-centred flow package is set with the `setConductivity` operation. The first argument is the layer type LCON flag specifying if the layer is either confined, unconfined or convertible, the second and third arguments are maps containing the horizontal and vertical hydraulic conductivity values for a layer. Transmissivity along rows is calculated automatically if the layer type is appropriate. The wetting capability is not used in this model.

The basic package is activated afterwards. The input map `bound.map` contains boundary condition values for each cell and is set with the `setBoundary` operation for a specific layer. Initial head values are set for each layer with the operation `setInitialHead`. The `setDIS` operation sets units, the number of timesteps within a stress period and the stress period to a transient simulation. The storage characteristics that are required for a transient simulation are set for each layer with the `setStorage` operation. Furthermore operations to set package specific options such as wetting iteration intervals are available. Using a solver, here the preconditioned conjugate-gradient package set with the `setPCG` operation, a model can be started with the `run` operation.

The supported head-dependent packages are the river and the drain package. The operation `setRiver` takes three input maps. The map `riv_l1h` holds the head values and `riv_l1b` contains the bottom elevation values. The map `riv_l1c` holds the hydraulic conductance of the riverbed. The `setDrain` operation activates the drain package and requires two input maps, `drn_elev` with the drain elevation values and `drn_c` with the conductance values. The last argument of both operations is the layer number the values are assigned to.

The head-independent well and recharge packages are specified with the operations `setWell` and `setRecharge` respectively. The `well.map` contains the pumping rates of the well located in the bottom layer. The `rch.map` holds the recharge values which are applied to the highest active cells.

The properties of the layer provided by the user are stored internally in a block structure. The values are used to create the several input files for MODFLOW. The resulting head and boundary values of a simulation are automatically imported and updated in the block structure of the MODFLOW extension object. The results of the packages can be obtained with `getHeads`, `getRiverLeakage`, `getRecharge` and `getDrain`. The result maps can be used in ongoing calculations or written to disk using the `report` operation.

4.2.3. Dynamic model section

Unlike the initial section, mainly consisting of links to the MODFLOW component, the dynamic section contains both native operations and operation calls to MODFLOW. Input data includes precipitation and reference evaporation data for one year. This data is used to calculate the input data for the recharge package on a daily basis as shown in line 2–7 of Table A6. The native `timeinputscalar` operations assign for each timestep precipitation and reference evaporation values to each cell in the catchment. The native `lookupscalar` operations assign to each cell a direct runoff value and a crop coefficient value (provided in the lookup tables `r.tbl` and `c.tbl`, respectively) to different landuse types given in the `landuse.map`. Those coefficients are used to calculate the runoff `ro` as a fraction of direct precipitation and the evapotranspiration as reference evapotranspiration weighted by the crop coefficient. The recharge finally applied to MODFLOW in line 7 is calculated as the effective precipitation diminished by the direct runoff and the evapotranspiration. The response of the head values to the incoming precipitation for a specific cell in the catchment is shown in Fig. A5.

The upward seepage calculated by the river package of MODFLOW is retrieved with two `getRiverLeakage` operations, each retrieving the seepage from a single layer number indicated by the function argument. By selecting the negative (i.e. upward) seepage

values only, using the native `max` operator, this results in a map `rivcmd` containing the total upward seepage for each cell.

The sum of the total upward seepage, direct runoff, and drainage discharge obtained from the drain package are used to calculate the total discharge of the stream "Langbroeker Wetering" (lines 12–16), see Fig. A4 for the location of the stream. The native `accuflux` operation calculates the total discharge (lateral flow) for each cell as the amount of material that is transported out of the cell. For each cell, this amount is composed of the material in the cell itself and the material from the upstream cells. Upstream cells are derived from the local drain direction map `ldd.map`. Here, the discharge is calculated by adding up the upward seepage, direct runoff and drainage from drains. Using three separate `accuflux` operations, it can be calculated for each individual discharge component, too. The results of this calculation are shown in Fig. A6.

5. Discussion and conclusion

In this paper we showed the necessity of integrating specialised model components into modelling frameworks for the development of large scale multicomponent models. We demonstrated the capabilities of two modelling languages to link external model components and applied this concept by means of constructing a groundwater model. We presented here the link to one external modelling component, while several components can be linked as well.

In general, the selection of a link strategy between modelling components depends on the type and quantity of the linked components. Fig. A7 shows three different groups of external model components. The groups shown in panel (a) and (b) of Fig. A7 can be linked to the modelling framework using the approach described in this paper. Fig. A7(a) depicts the situation of linking model components without time dimension or components representing a time interval bounded by the length of the framework timestep interval. Such external components receive as input the current state of the framework relevant for the component, parameters, and if required a time interval predetermined by the modelling framework. The component calculates the new state related to the component and returns it to the modelling framework where it is stored or adjusted to serve as input to the external component in the next timestep. As components are executed each timestep and no bookkeeping of component states is necessary, this kind of linking is adequate for single operations or smaller components. An example for this approach is the *colMedian* operation described in Section 3.1.2.

Fig. A7(b) shows the second group, where bigger components with timeslices independent from the framework or components keeping their states beyond time intervals given by the framework are linked. An example of this group is the MODFLOW component described here. The architecture of MODFLOW allows to separate its runtime into stress periods which can be mapped to the framework timesteps. Additionally to the framework state the component state must be stored either by the framework or the component. Here the state is updated each timestep in the MODFLOW extension object. This approach is convenient for MODFLOW because it uses the stress periods. It may also be applicable for other model components that belong to the second group, but only if their architecture provides a means to map the runtime to the framework timesteps.

The approaches of linking presented here integrate components into the modelling framework and thus allow the model developer to use various components in a single modelling framework. Linking external components hereby does not only enrich the framework power of expression, but can also be used to ease acquaintance with external components. In our case, with only using the extension operations, one is for instance able to use the PCRaster framework as free pre- and postprocessor for MODFLOW. However, for both groups (Fig. A7a and b) of external model components, the runtime of the linked components must be separable into timeslices matching the ones given by the modelling frameworks, and administration of states has to be considered. Hence this technique is practicable for a small number of components, but becomes more difficult when a large number of components need to be linked. This increases the complexity because administration and data transfer for a large number of component model states need to be done, while conformity needs to be kept of the spatial and temporal discretisation of components.

For a large number of components a different approach is more suitable. Fig. A7(c) shows a situation of integrated modelling that is difficult to handle with the linking approach described in this paper. In this situation, a large number of model components need to be integrated, where it is also difficult to map their timesteps to the timesteps of the modelling framework. Here, it is better to use a modelling interface such as OpenMI (2008). OpenMI is a communication standard for linking modelling frameworks. Each participating model has to specify its input and output requirements and data transfer and time flow are organised by OpenMI. This specification eases the execution of different modelling components. Newly developed modelling components following a reusable and modular design (Rizzoli and Argent, 2006) can be made OpenMI compliant with minor efforts. As OpenMI is a recent approach originated in Europe not all model components comply to the standard, or are as stand-alone applications easily convertible. The drawback of our approach, the close-coupled link to PCRaster, is diminished by the fact that OpenMI compliance of PCRcalc is currently under development.

We also demonstrated how the two modelling languages can be utilised to build integrated models. The tailored language PCRcalc is appropriate for model developers without any programming experience. Provision of operations targeted to environmental model developers with a readable syntax allows a straightened model development. This is advantageous for instance in teaching, where the emphasis is on model development and replicating environmental processes instead of programming. By contrast, Python, being a general purpose programming language, requires a basic proficiency of the language itself, for example error messages can be less understandable than given in the tailored language. In return, the model developer can use Python language features and has the option to introduce own data types or include modules developed by external groups (see also Karssenberg et al., 2007). Furthermore, the object-oriented Python language enforces a modular design of a model structure which means partitioning sub-components of a model into different Python modules. This results in a higher maintainability for each component and the complete model, a potential for a re-use of sub-components and a better overview of complex models.

The framework presented in this paper allows a model developer to construct integrated models without specialist knowledge of low-level programming. A number of other approaches exist to construct this type of model (Karssenberg, 2002). One approach is to use other existing toolsets, but the model development process with these toolsets may be subject to restrictions. Geographic Information Systems have their focal point in spatial analysis but do not provide sophisticated capabilities for dynamic modelling. Storage based modelling toolsets like STELLA (2008) allow to model temporal dependencies, but do not support to represent spatial interactions and processes. MATLAB (2005) has its main field of application in the engineering domain, but can be used to perform analysis on environmental data. However, spatial and temporal operations are not natively supported and need to be constructed. In contrast, the PCRaster framework provides performance

optimised operations suitable for spatio-temporal model development in the field of environmental and earth sciences. Furthermore, exploratory data analysis is assisted by the framework visualisation tool that enables prompt visualisation of spatial, temporal or stochastic data (Pebesma et al., 2007). In addition, the framework bindings to the object-oriented Python programming language allow a modular design which is advantageous for structured model development (Rizzoli and Argent, 2006).

Specialised toolsets simulating a specific component or process in the environment, for instance VisualMODFLOW (2008) that uses the MODFLOW modelling engine to model groundwater flow, are not efficient tools for integrated modelling of a large environmental system. The reason therefore is that interfaces to link them to other modelling tools are mostly not available and need to be developed from scratch in each project.

An implementation of a model in a system programming language like Fortran, C or C++ allows integrated dynamic modelling, but model construction is difficult: spatial data types, a temporal framework, visualisation software and more must be developed from scratch. This demands specialist programming knowledge, is time consuming, error-prone and in most cases beyond the scope of a model developer.

While the main focus of the framework is on the development of research models it can also play a beneficial role in the development of integrated models for different end-users like policy makers. The framework offers a unified interface to various component models. This eases for instance the development of a user-friendly interface on top of the framework that enables the end-user to specify model input scenarios (Oxley et al., 2002).

Acknowledgements

We acknowledge the financial support by the Dutch research programme Space for Geo-Information (Ruimte voor Geo-Informatie, RGI) as RGI project 228 "The OpenMI framework for access to spatial temporal data and the linking of models". Furthermore we thank Kor de Jong for developing components of the PCRaster Python extension and valuable remarks. Arien Lam is acknowledged for his feedback during the extension development. Finally, we thank three anonymous reviewers for their useful comments to the manuscript.

Appendix A. PCRaster Python example script

The following code shows the Python version of the PCRcalc script given in the case study of Section 4. The *binding*, *areamap* and *timer* sections are no longer necessary. Names and argument types of the PCRasterModflow operations equal the PCRcalc version.

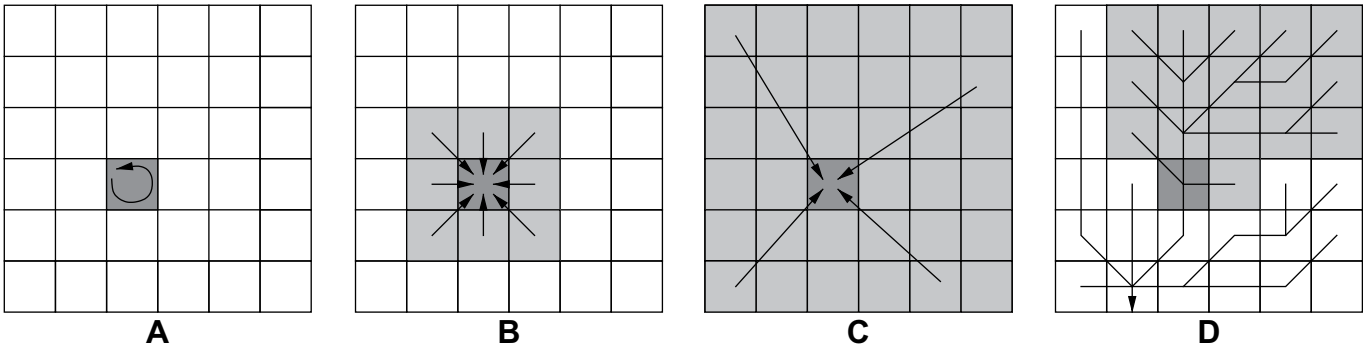

Fig. A1. Classification of operations: (A) Point or local, (B) Direct neighbourhood or focal, (C) Entire neighbourhood or zonal, and (D) Functions with a neighbourhood defined by a given topology.

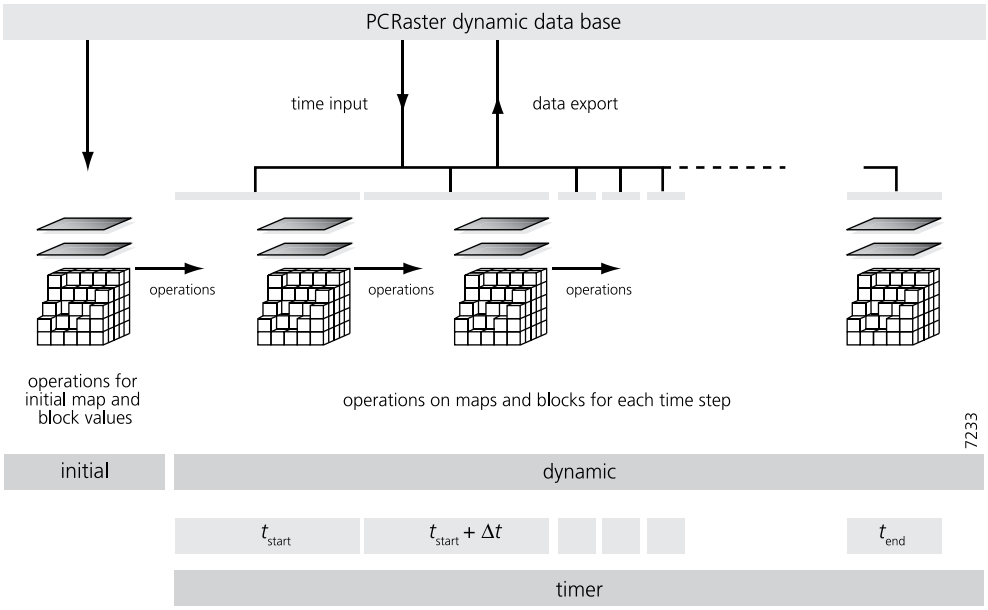

Fig. A2. Conceptual overview of a PCRaster model run. The initial section is used to establish valid starting conditions, the dynamic section performs the process descriptions for each timestep.

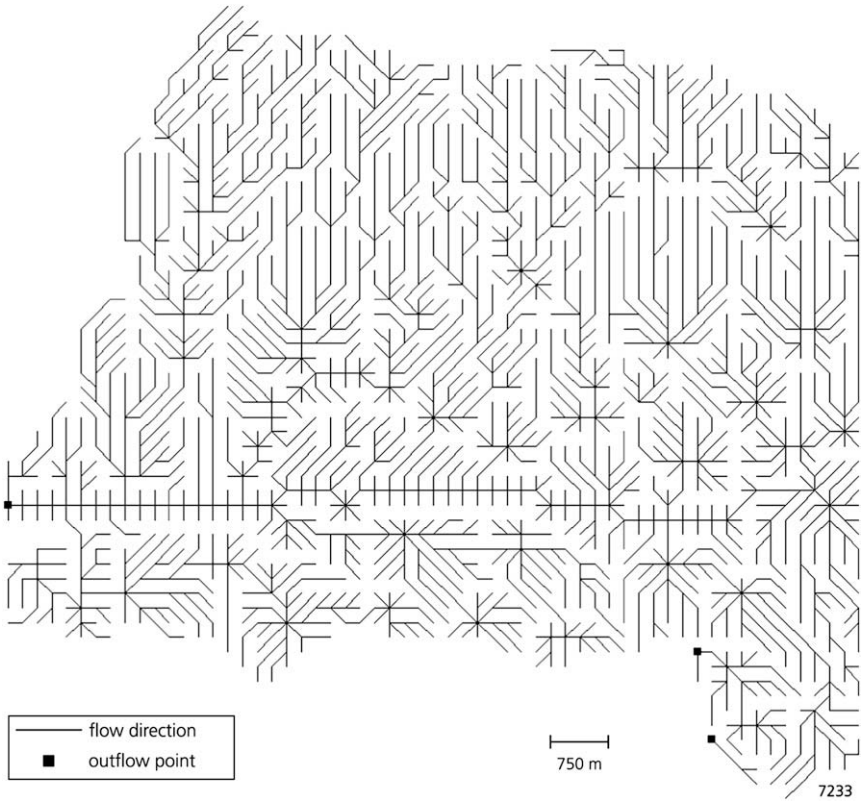

Fig. A3. Local drain direction network map of a catchment. Flow directions are indicated from each cell to its steepest downslope neighbour. The outlet is located at the left side of the catchment.

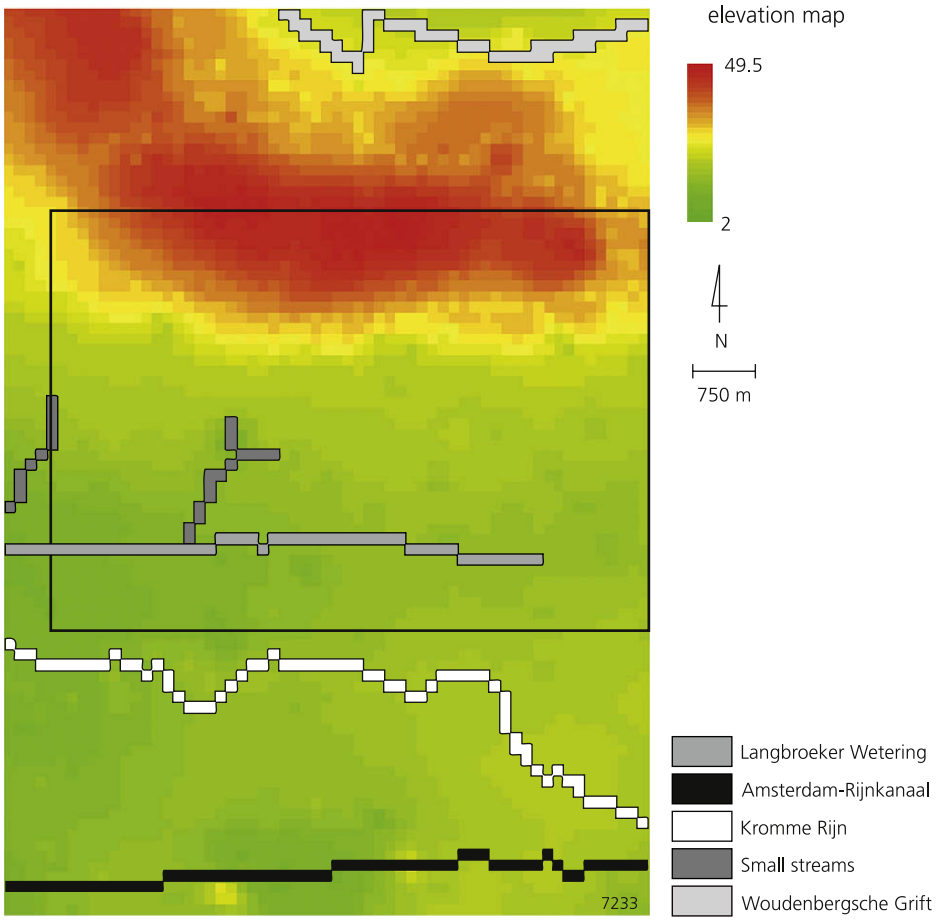

Fig. A4. Map showing elevation values and streams. The major stream in the south is the Amsterdam–Rhine canal with an average width of 150 m, the smaller streams have an average width of 10 m. The box indicates the location of the basin shown in Fig. A3.

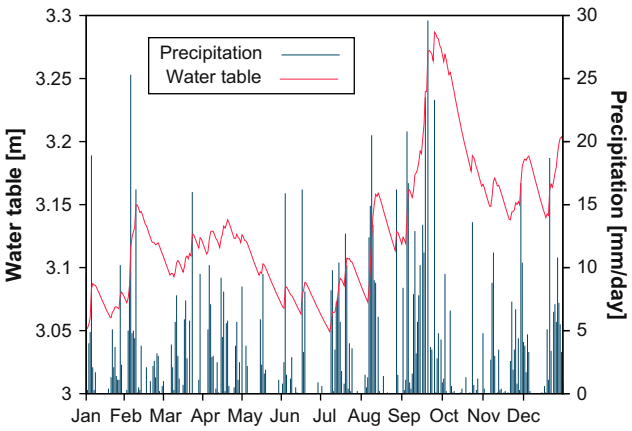

Fig. A5. Response of the water table to the effective precipitation for one specific cell in the catchment.

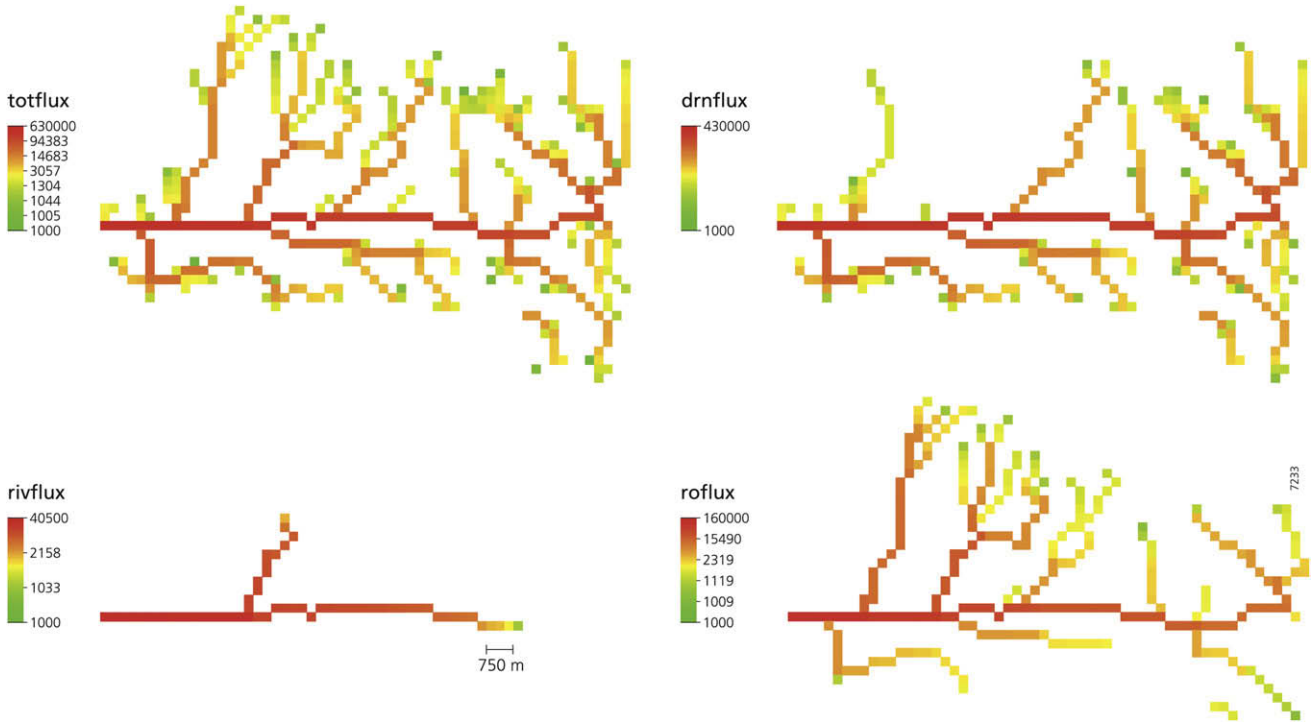

Fig. A6. Simulated total and separate fluxes in a subcatchment for September, 15 (timestep 258) after a strong rainfall event. Units in m^3/day.

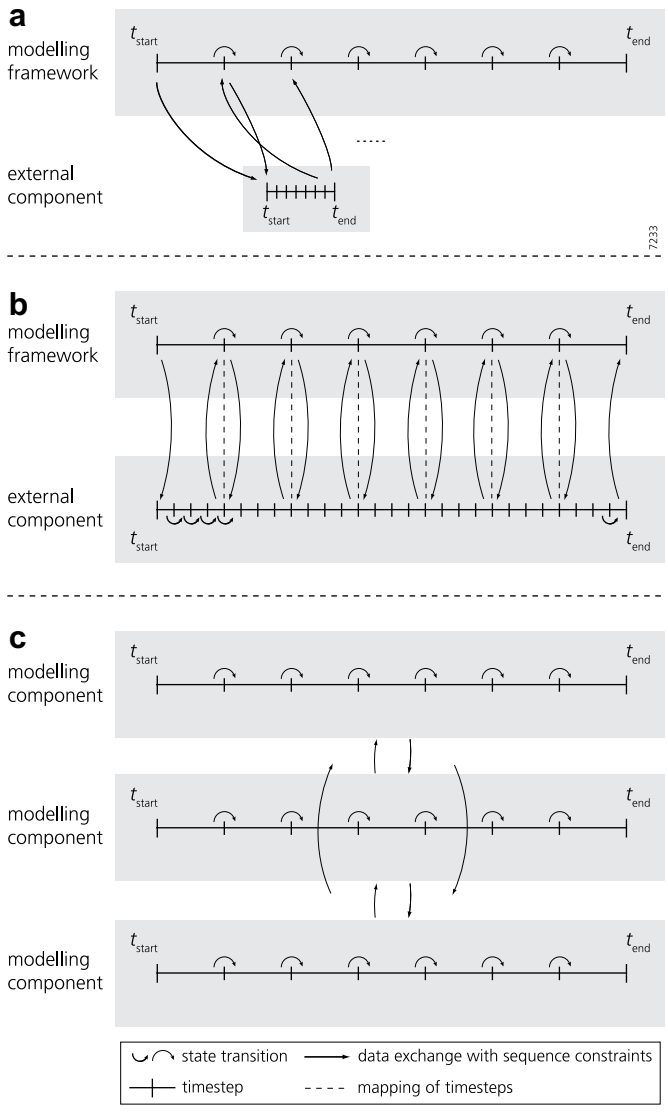

Fig. A7. Different approaches of linking model components. (a) Linking a single operation (b) Link to an independent modelling component, and (c) Arbitrary linking of modelling components.

Table A1
PCRcalc model script simulating surface runoff for one year.

```
binding
  dem = dem.map;
areamap
  clone.map;
timer
  1 52 1;
initial
  ldd = lddcreate(dem, 1E31, 1E31, 1E31, 1E31);
dynamic
  precip = timeinputscalar(rain.tss, rainzone.map);
  report runoff = accuflux(ldd, precip);
```

Table A2
XML manifest for an extension library specifying a single operation named *colMedian*.

```
<?xml version="1.0"?>
<linkInLibraryManifest xmlns="...">
 <function>
  <name>colMedian</name>
  <result>
   <dataType><scalar/></dataType>
   <spatialType>Spatial</spatialType>
  </result>
  <argument>
   <dataType><scalar/></dataType>
   <spatialType>Spatial</spatialType>
  </argument>
 </function>
</linkInLibraryManifest>
```

Table A3
C++ code extract from the extension library. PCRaster data types are exposed to the extension library which executes calculations on the map values.

```
DLL_FUNC (char const *) pcr_LinkInExecute(
      char const *xml,
      LinkInTransferArray transferArray){
  float *result = (float*)transferArray[0];
  float *input = (float*)transferArray[1];
  /* calculate new values */
  return 0;
}
```

Table A4
The surface runoff model in the PCRaster Python language.

```
from pcraster import *

class RunoffModel(object):
  def__init__(self, cloneMap):
    setclone(cloneMap)

  def initial(self):
    self.ldd = lddcreate("dem.map", 1E31, 1E31, 1E31, 1E31)

  def dynamic(self):
    precip = timeinputscalar("rain.tss", "rainzone.map")
    ro = accuflux(self.ldd, precip)
    report(ro, "runoff")

DynamicFramework(RunoffModel("clone.map"), 52).run()
```

Table A5
Catchment model script in the tailored PCRcalc language up to and including the initial section. Identical commands for layer 2 are omitted.

```
binding
  boundaries = bound.map;
areamap
  clone.map;
timer
  1 365 1;
initial
  object mf = PCRasterModflow::initialise();
  # defining the thickness of the layer
  mf::createBottomLayer(bottom.map, l1_top.map);
  mf::addLayer(elev.map);
  # hydraulic conductivity
  mf::setConductivity(0, l1_k.map, l1_k.map, 1);
  # boundary conditions and starting values
  mf::setBoundary(boundaries, 1);
  mf::setInitialHead(iHead.map, 1);
  mf::setStorage(storage.map, storage.map, 1);
  # simulation parameter and solver package
  mf::setDISParameter(4, 2, 1, 24, 1, 0);
  mf::setPCG(50, 30, 1, 0.001, 0.001, 1.0, 2, 1);
  # river package
  mf::setRiver(riv_l1h.map, riv_l1b.map, riv_l1c.map, 1);
  # drains in the top layer
  mf::setDrain(drn_elev.map, drn_c.map, 2);
  # single well, located in the bottom layer
  mf::setWell(well.map, 1);
```

Table A6
Model script continued. Discharge of the streams is composed of surface runoff, river and drain leakage and simulated for one year.

```
dynamic
 p = timeinputscalar(precip.tss, clone.map) * 0.001; # mm to m
 e = timeinputscalar(evapo.tss, clone.map) * 0.001; # mm to m
 ro = lookupscalar(r.tbl, landuse.map) * p;
 e = lookupscalar(c.tbl, landuse.map) * e;
 # applying recharge to highest active cell
 mf::setRecharge(p – ro – e, 3);
 # calling MODFLOW executable
 mf::run();
 hOne = mf::getHeads(1);
 # river leakage
 rivtot = mf::getRiverLeakage(2) + mf::getRiverLeakage(1);
 # calculating upward seapage
 rivcmd = max(0.0 – rivtot, 0.0);
 # flux composed of runoff, river leakage and drain leakage
 report totflux = accuflux(ldd.map, ro * 22500  +rivcmd + mf::getDrain(1));
```

Table A7
Integrated catchment model written in the PCRaster Python language. Identical commands for layer 2 are omitted.

```
from pcraster import *
from PCRasterModflow import *

class CatchmentModel(object):
 def __init__(self, cloneMap):
  setclone(cloneMap)

 def initial(self):
  self.mf = PCRasterModflow(clone())
  # defining the thickness of the layer
  self.mf.createBottomLayer("bottom.map", "l1_top.map")
  self.mf.addLayer("elev.map")
  # hydraulic conductivity
  self.mf.setConductivity(0, "l1_k.map", "l1_k.map", 1)
  # boundary conditions and starting values
  self.mf.setBoundary("bound.map", 1)
  self.mf.setInitialHead("iHead.map", 1)
  self.mf.setStorage("storage.map", "storage.map", 1)
  # simulation parameter and solver package
  self.mf.setDISParameter(4, 2, 1, 24, 1, 0)
  self.mf.setPCG(50, 30, 1, 0.001, 0.001, 1.0, 2, 1)
  # river package
  self.mf.setRiver("riv_l1h.map","riv_l1b.map","riv_l1c.map",1)
  # drains in the top layer
  self.mf.setDrain("drn_elev.map", "drn_c.map", 2)
  # single well, located in the bottom layer
  self.mf.setWell("well.map", 1)

 def dynamic(self):
  p = timeinputscalar("precip.tss","clone.map") * 0.001 # mm to m
  e = timeinputscalar("evapo.tss","clone.map") * 0.001 # mm to m
  ro = lookupscalar("r.tbl", "landuse.map") * p
  e = lookupscalar("c.tbl", "landuse.map") * e
  # applying recharge to highest active cell
  self.mf.setRecharge(p – ro – e, 3)
  # calling MODFLOW executable
  self.mf.run()
  hOne = self.mf.getHeads(1)
  # river leakage
  rivtot = self.mf.getRiverLeakage(2)+self.mf.getRiverLeakage(1)
  # calculating upward seepage
  rivcmd = max(0.0 – rivtot, 0.0)
  # flux composed of runoff, river leakage and drain leakage
  report(accuflux("ldd.map", ro * 22500
          +rivcmd + self.mf.getDrain(1),"totflux")

myModel = CatchmentModel("clone.map")
dynModelFw = DynamicFramework(myModel, 365)
dynModelFw.run()
```

References

ArcGIS, 2008. 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.

Batelaan, O., De Smedt, F., 2004. Seepage, a new Modflow Drain package. Ground Water 42 (4), 576–588.

Beck, M., Jakeman, A., McAleer, M., 1993. Construction and evaluation of models of environmental systems. In: Beck, M., Jakeman, A., McAleer, M. (Eds.), Modelling Change in Environmental Systems. John Wiley & Sons Ltd., New York, pp. 3–35.

Benenson, I., Torrens, P.M., 2004. Geosimulation: Automata-Based Modeling of Urban Phenomena. Wiley, Chichester.

Boost Python, 2008. Boost C++ libraries. URL: http://www.boost.org/libs/python/.

Burrough, P.A., Karssenberg, D., van Deursen, W.P., 2005. Environmental modelling with PCRaster. In: Maguire, D., Goodchild, M.F., Batty, M. (Eds.), GIS, Spatial Analysis and Modeling. ESRI, Redlands, California.

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

Carrera-Hernández, J.J., Gaskin, S.J., 2006. The groundwater modeling tool for GRASS (GMTG): open source groundwater flow modeling. Computers & Geosciences 32, 339–351.

De Vasconcelos, M.J.P., Goncalves, A., Catry, F.X., Paul, J.U., Barros, F., 2002. A working prototype of a dynamic geographical information system. International Journal of Geographical Information Science 16 (1), 69–91.

Gedeon, M., Wemaere, I., Marivoet, J., 2007. Regional groundwater model of northeast Belgium. Journal of Hydrology 335 (1–2), 133–139.

GRASS, 2008. Geographic Resources Analysis Support System. URL: http://grass.itc.it/.

Gregersen, J.B., Gijsbers, P.J., Westen, S.J., 2007. OpenMI: Open Modelling Interface. Journal of Hydroinformatics 9 (3), 175–191.

Grimm, V., Railsback, S., 2005. Individual-Based Modelling and Ecology. Princeton University Press, Princeton.

Harbaugh, A., Banta, E., Hill, M., McDonald, M., 2000. MODFLOW-2000, the US Geological Survey Modular Ground-Water Model – User Guide to Modularization Concepts and the Ground-Water Flow Process. U.S. Geological Survey.

Herzog, B.L., Larson, D.R., Abert, C.C., Wilson, S.D., Roadcap, G.S., 2003. Hydrostratigraphic modeling of a complex, glacial-drift aquifer system for importation into MODFLOW. Ground Water 41 (1), 57–65.

Hurkens, J., Hahn, B., van Delden, H., 2008. Using the GEONAMICA software environment for integrated dynamic spatial modelling. In: Sánchez-Marrè, M., Béjar, J., Comas, J., Rizzoli, A., Guariso, G. (Eds.), Integrating Sciences and Information Technology for Environmental Assessment and Decision Making iEMSs 2008: International Congress on Environmental Modelling and Software.

Karssenberg, D., Bridge, J.S., 2008. A three-dimensional numerical model of sediment transport, erosion and deposition within a network of channel belts, floodplain and hill slope: extrinsic and intrinsic controls on floodplain dynamics and alluvial architecture. Sedimentology 55, 1717–1745.

Karssenberg, D., Burrough, P., Sluiter, R., de Jong, K., 2001. The PCRaster software and course materials for teaching numerical modelling in the environmental sciences. Transactions in GIS 5, 99–110.

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., Schmitz, O., 2008a. A software environment for stochastic spatio-temporal modelling. In: European Geosciences Union, vol. 10. EGU General Assembly, Wien.

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., de Vries, L., de Jong, K., 2008b. A tool for construction of stochastic spatio-temporal models assimilated with observational data. In: Bernard, L., Friss-Christensen, A., Pundt, H., Compte, I. (Eds.), 11th AGILE International Conference on Geographic Information Science Girona. URL: http://www.agile-online.org/.

Karssenberg, D., 2002. The value of environmental modelling languages for building distributed hydrological models. Hydrological Processes 16 (14), 2751–2766.

Leavesley, G., Restrepo, P., Markstrom, S., Dixon, M., Stannard, L., 1998. The Modular Modeling System – MMS: User's Manual, vol. 96–151 U.S. Geological Survey Open File Report.

MATLAB, 2005. MATLAB product website. URL: http://www.mathworks.com/.

McDonald, M., Harbaugh, A., 1984. A Modular Three-Dimensional User's Guide for Finite-Difference Ground-Water Flow Model, vol. 94–464 USGS OFR.

McIntosh, B.S., Jeffrey, P., Lemon, M., Winder, N., 2005. On the design of computer-based models for integrated environmental science. Environmental Management 35 (6), 741–752.

McIntosh, B.S., Seaton, R.A.F., Jeffrey, P., 2007. Tools to think with? Towards understanding the use of computer-based support tools in policy relevant research. Environmental Modelling & Software 22 (5), 640–648.

Nguyen, C.D., Araki, H., Yamanishi, H., Koga, K., 2005. Simulation of groundwater flow and environmental effects resulting from pumping. Environmental Geology 47 (3), 361–374.

OpenMI, 2008. Open Modelling Interface website. URL: http://www.openmi.org/.
Osman, Y.Z., Bruen, M.P., 2002. Modelling stream – aquifer seepage in an alluvial aquifer: an improved loosing-stream package for MODFLOW. Journal of Hydrology 264 (1–4), 69–86.
Oxley, T., Jeffrey, P., Lemon, M., 2002. Policy relevant modelling: relationships between water, land use and farmer decision processes. Integrated Assessment 3, 30–49.
Oxley, T., McIntosh, B.S., Winder, N., Mulligan, M., Engelen, G., 2004. Integrated modelling and decision-support tools: a Mediterranean example. Environmental Modelling & Software 19, 999–1010.
PCRaster, 2008. PCRaster internet site. URL: http://pcraster.geo.uu.nl.
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.
Processing Modflow, 2008. Processing Modflow website. URL: http://www.pmwin.net/.
Pullar, D., 2003. Simulation modelling applied to runoff modelling using MapScript. Tranactions in GIS 7 (2), 267–283.
Pullar, D., 2004. Simumap: a computational system for spatial modelling. Environmental Modelling & Software 19 (3), 235–243.
Python, 2008. Python programming language website. URL: http://www.python.org/.
Repast, 2008. Recursive Porous Agent Simulation Toolkit Website URL: http://repast.sourceforge.net/.
Rizzoli, A.E., Argent, R.M., 2006. Software and software systems: platforms and issues for IWRM Problems. In: Sustainable Management of Water Resources: an Integrated Approach. Edward Elgar Publishing, pp. 324–346.
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.
STELLA, 2008. STELLA product website. URL: http://www.iseesystems.com/.
Tomlin, C.D., 1990. Geographic Information Systems and Cartographic Modeling. Prentice Hall.
USGS, 2008. US Geologic Survey. URL: http://water.usgs.gov/nrp/gwsoftware/modflow.html.
van der Knijff, J.M., Younis, J., de Roo, A.P.J., 2008. LISFLOOD: a GIS-based distributed model for river basin scale water balance and flood simulation. International Journal of Geographical Information Science. doi:10.1080/13658810802549154.
van Deursen, W., 1995. Geographical Information Systems and Dynamic Models. Koninklijk Nederlands Aardrijkskundig Genootschap/Faculteit Ruimtelijke Wetenschappen, Universiteit Utrecht, Utrecht.
Visser, S., Sterk, G., Karssenberg, D., 2004. Wind erosion modelling in a Sahelian environment. Environmental Modelling & Software 20, 69–84.
VisualMODFLOW, 2008. Visual Modflow product website. URL: http://www.visualmodflow.com/.
Wainwright, J., Mulligan, M., 2004. Environmental Modelling. Wiley, Chichester.
Wang, X., Pullar, D., 2005. Describing dynamic modeling for landscapes with vector map algebra in GIS. Computers & Geosciences 31 (8), 956–967.
Wesseling, C., Karssenberg, D., van Deursen, W., Burrough, P., 1996. Integrating dynamic environmental models in GIS: the development of a Dynamic Modelling language. Transactions in GIS 1, 40–48.
Winston, R.B., 1999. Modflow-related freeware and shareware resources on the internet. Computers & Geosciences 25 (4), 377–382.