Integrating environmental component models
Development of a software framework

Integratie van component modellen
Ontwikkeling van een software raamwerk voor het maken van landschapsmodellen

(met een samenvatting in het Nederlands)

P R O E F S C H R I F T

ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof. dr. G. J. van der Zwaan,
ingevolge het besluit van het college voor promoties in het openbaar te verdedigen

op donderdag 8 mei 2014 des middags te 4.15 uur

door

Oliver Schmitz

geboren op 27 maart 1975 te Husum, Duitsland

Promotoren:
Prof. dr. S. M. de Jong
Prof. dr. ir. M. F. P. Bierkens

Copromotoren:
Dr. D. Karssenberg
Dr. J.-L. de Kok

Integrating environmental component models

Development of a software framework

Utrecht Studies in Earth Sciences

ISSN 2211-4335

Utrecht Studies in Earth Sciences 53

Integrating environmental component models
Development of a software framework

Oliver Schmitz

Utrecht 2014

Department Physical Geography
Faculty of Geosciences - Utrecht University

Promotoren:
Prof. dr. S. M. de Jong
Prof. dr. ir. M. F. P. Bierkens

Copromotoren:
Dr. D. Karssenberg
Dr. J.-L. de Kok

Examination committee
Prof. dr. ir. Arnold K. Bregt, Wageningen University
Prof. dr. Edzer J. Pebesma, University of Münster, Germany
Prof. dr. Dimitri Solomatine, UNESCO–IHE
Prof. dr. Jantien E. Stoter, Delft University of Technology
Dr. Andrea E. Rizzoli, IDSIA, Switzerland

This work was financially supported by the Flemish Institute for Technological Research (VITO, Belgium) under contract RMA N8104.

ISBN 978-90-6266-355-2

Printed in the Netherlands by CPI Wöhrmann Print Service, Zutphen

All these languages influenced our work,
but it was more fun to do things on our own.

Dennis M. Ritchie

Contents

Abstract

Integrated models consist of interacting component models that represent various natural and social systems. They are important tools to improve our understanding of environmental systems, to evaluate cause–effect relationships of human–natural interactions, and to forecast the behaviour of environmental systems. The construction of these models is a conceptual and technical challenge, as it requires integrating various environmental processes, often occurring at different spatial or temporal scales. Several software tools are available supporting the development of integrated models but their diversity and IT–orientation hampers domain specialists such as hydrologists in the construction and assessment of integrated models.

This dissertation contributes to the development of one modelling framework allowing for the construction of component models, their multi–scale coupling to integrated models, and the analysis thereof. We consider different aspects of modelling frameworks including model syntax and execution, spatial and temporal scaling, integration of legacy code, support for model analysis and semantic interoperability. We propose solutions to these issues tailored to domain specialists.

Model construction and evaluation becomes difficult for domain specialists when technical barriers are present. We show that operations implementing domain specific concepts can be used by modellers to express spatio–temporal processes in component models, and they can be used to express couplings and therefore feedback effects between component models. To execute models described with domain specific concepts, we analyse with a client–server and a request–reply approach two different control flow mechanisms for component models and integrated models. Overall, both approaches allow for the sound execution of multi–scale integrated models. The client–server approach allows for better optimisation of the model execution due to the overall knowledge of the model.

By introducing the accumulator as a generic model building block of a modelling framework, we provide the modeller a means to program scale transfer operations on space and time with operations representing domain concepts similar to map algebra operations. The accumulators allow for a coupling of multi–scale components without the need to modify process implementations and spatial or temporal characteristics of individual component models.

We present two ways to integrate existing model applications into the modelling frameworks, allowing therefore to reuse functionality from a large base of available legacy models. The integration of external models done by software engineers on the level of Application Programming Interfaces provides modellers functionality in the same modelling language as components are constructed and offers performance benefits. On the other hand, the integration of applications by system calls is easier to realise for domain specialists.

The numerical implementation representing environmental processes does not necessarily expose the scientific meaning of component models. We develop a formal

description of the main model building blocks allowing for a semantically enhanced description of spatial and temporal characteristics of these building blocks, and evaluate whether an implementation–independent formalisation improves the scientific interoperability.

We demonstrate that the presented concepts for component construction and multi–scale coupling can be merged into one modelling platform. Software framework prototypes are used in illustrative case studies, where we construct component models with spatio–temporal processes simulating amongst others land use change, hydrological processes, or biomass growth, and couple these to integrated models.

Samenvatting

Geïntegreerde modellen bestaan uit met elkaar communicerende componentmodellen, die verschillende natuurlijke en sociale systemen representeren. Het zijn belangrijke instrumenten om ons begrip van ruimtelijke systemen te verbeteren, om causale verbanden tussen menselijke en het natuurlijke systeem te evalueren en om het gedrag van ruimtelijke systemen te voorspellen. De bouw van geïntegreerde modellen is een conceptuele en technische uitdaging, omdat het vereist dat verschillende processen worden geïntegreerd, die vaak op verschillende ruimtelijke en temporele schalen spelen. Verschillende software tools zijn beschikbaar om de ontwikkeling van geïntegreerde modellen te ondersteunen, maar hun diversiteit en ICT–oriëntatie belemmeren domeinspecialisten, zoals hydrologen, in de bouw en evaluatie van geïntegreerde modellen.

Dit proefschrift draagt bij aan de ontwikkeling van een software raamwerk waarmee de bouw van componentmodellen en de koppeling ervan tot geïntegreerde modellen mogelijk wordt. Daarnaast ondersteunt het software raamwerk de analyse van modeluitkomsten. We beschouwen verschillende aspecten van een software raamwerk, met name de syntax van een model script, het uitvoeren van een model op de computer, de ruimtelijke en temporele schaling, de integratie van bestaande modelcode, de ondersteuning van modelanalyse en de semantische interoperabiliteit. We stellen oplossingen voor die zijn afgestemd op domeinspecialisten.

Op dit moment zijn er technische belemmeringen die domeinspecialisten hinderen bij de bouw en evaluatie van modellen. Om deze belemmeringen op te heffen, ontwikkelen we model operaties gebaseerd op domeinspecifieke concepten. Deze operaties kunnen modelbouwers ten eerste gebruiken om temporele en ruimtelijke processen in componentmodellen te beschrijven en ten tweede om koppelingen (en terugkoppelingen) tussen componentmodellen te definiëren. We analyseren twee verschillende control–flow mechanismen om geïntegreerde modellen uit te voeren op een computer, een client–server en een request–reply aanpak. Over het algemeen kunnen met beide benaderingen geïntegreerde modellen correct worden uitgevoerd. De client–server aanpak heeft echter als voordeel dat voorafgaand aan het uitvoeren van het model kennis wordt verzameld over de structuur van het model, waardoor betere optimalisatie mogelijk wordt bij het uitvoeren van het model.

Door de introductie van de accumulator als een generieke modelbouwsteen van het software raamwerk bieden wij de modelbouwer een middel om schalende operaties op ruimte en tijd te implementeren met operaties gebaseerd op domeinconcepten vergelijkbaar met het bestaande ‘map algebra’ concept. De accumulatoren faciliteren de koppeling van componentmodellen die op verschillende schalen berekeningen uitvoeren zonder dat de omschrijving van de processen en ruimtelijke of temporele kenmerken van individuele componentmodellen hoeft te worden gewijzigd.

We presenteren twee manieren om bestaande modellen in het software raamwerk te integreren, waardoor functionaliteit van een grote groep van beschikbare modellen hergebruikt kan worden. De integratie van externe modellen, uitgevoerd door software-

ontwikkelaars op het niveau van Application Programming Interfaces, biedt modelbouwers functionaliteit in dezelfde modelleertaal als waarin de componentmodellen worden gebouwd en resulteert in betere prestaties. De integratie van applicaties door 'system calls' daarentegen is makkelijker te realiseren door domeinspecialisten.

Om koppeling van componentmodellen mogelijk te maken zonder dat kennis van de interne werking van componentmodellen vereist is, ontwikkelen we een methode om componentmodellen formeel te beschrijven. Dit resulteert in een betere semantische beschrijving van de ruimtelijke en temporele kenmerken van componentmodellen, wat de interoperabiliteit sterk verbetert.

We laten zien dat de gepresenteerde concepten voor het bouwen en koppelen van componentmodellen kunnen worden omgezet in een software prototype. We illustreren de werking van deze prototype software met behulp van een aantal casussen, waarin componentmodellen worden gebouwd die ruimtelijke en temporele processen zoals landgebruiksverandering, hydrologische processen of de groei van biomassa simuleren. Tevens worden deze componentmodellen gekoppeld tot geïntegreerde modellen.

Zusammenfassung

Integrierte Modelle bestehen aus miteinander kommunizierenden Komponentenmodellen, die unterschiedliche Prozesse von natürlichen und sozialen Systemen repräsentieren. Sie sind wichtige Werkzeuge, um unser Verständnis von räumlichen Systemen zu verbessern, Ursache–Wirkung–Beziehungen resultierend aus Interaktionen zwischen Mensch und Natur zu bewerten, und das Verhalten von natürlichen Systemen zu prognostizieren. Die Entwicklung integrierter Modelle ist eine konzeptuelle und technische Herausforderung, da eine Vielzahl von Prozessen zu integrieren sind, die oftmals auf unterschiedlichen räumlichen und zeitlichen Skalen agieren. Software, die die Entwicklung von integrierten Modellen unterstützt, steht zwar zur Verfügung, deren Vielfalt und informationstechnische Ausrichtung allerdings erschwert Wissenschaftlern unterschiedlicher Disziplinen, wie zum Beispiel Hydrologen, die Entwicklung und Auswertung von integrierten Modellen.

Diese Dissertation trägt zur Entwicklung eines Softwareframeworks bei, mit der die Entwicklung von Komponentenmodellen, ihre Zusammensetzung bei unterschiedlichen Skalen zu integrierten Modellen und deren Analyse möglich ist. Wir betrachten dabei unterschiedliche Aspekte von Softwareframeworks, einschließlich der Modellsyntax und Modellausführung, der räumlichen und zeitlichen Skalierung, der Integration von bestehenden Modellen, der Unterstützung für Analyse von Modellen und der semantischen Interoperabilität. Wir präsentieren Lösungen für diese Probleme, die auf die Bedürfnisse von Forschern der Umweltwissenschaften zugeschnitten sind.

Technische Hindernisse behindern Wissenschaftler derzeit in der unkomplizierten Entwicklung und Auswertung von Modellen. Wir zeigen, daß Operationen, die domänenspezifische Konzepte umsetzen, durch Modellierer verwendet werden können um räumliche und zeitliche Prozesse in Komponentenmodellen zu beschreiben. Sie können ebenfalls verwendet werden, um Verbindungen und somit Feedbackeffekte zwischen Komponentenmodellen auszudrücken. Um Modelle, die mit domänenspezifischen Konzepten beschrieben sind, auszuführen, analysieren wir mit einem Client–Server–Ansatz und einem Request–Reply–Ansatz zwei verschiedene Kontrollflußmechanismen für die Ausführung von sowohl Komponentenmodellen als auch integrierten Modellen. Insgesamt ermöglichen beide Ansätze die korrekte Ausführung von Modellen. Der Client–Server–Ansatz ermöglicht eine bessere Optimierung der Modellausführung aufgrund der umfassenden Kenntnis eines Modells.

Durch das Bereitstellen des Akkumulators als ein generischer Modellbaustein eines Softwareframeworks geben wir dem Modellierer ein Mittel, räumliche und zeitliche Skalierungen mit Operationen zu entwickeln, die vergleichbar zur Map Algebra sind. Die Akkumulatoren ermöglichen eine Verbindung von Komponentenmodellen ohne die Notwendigkeit, Prozeßbeschreibungen und räumliche oder zeitliche Eigenschaften der einzelnen Komponentenmodelle zu ändern.

Wir präsentieren zwei Möglichkeiten, um bestehende Modelle in ein Softwareframework zu integrieren, so daß vorhandene Funktionalität von bestehenden Modellen

wiederverwendet werden kann. Die Integration externer Modelle, ausgeführt von Softwareentwicklern auf der Ebene von Application Programming Interfaces, bietet den Modellierern die gewünschte Funktionalität in der gleichen Modellierungssprache, in der auch Komponenten beschrieben sind, und sie bietet Vorteile im Laufzeitverhalten. Andererseits ist die Integration von bestehenden Modellen durch Systemaufrufe für Modellierer einfacher zu realisieren.

Eine numerische Prozeßbeschreibung gibt nicht zwangsläufig die wissenschaftliche Bedeutung eines Komponentenmodells wieder. Wir entwickeln eine Formalisierung für die wichtigsten Modellbausteine, die eine semantisch erweiterte Beschreibung der räumlichen und zeitlichen Eigenschaften dieser Bausteine ermöglicht. Darüber hinaus beurteilen wir, ob eine Formalisierung unabhängig von der Implementierung die wissenschaftliche Interoperabilität von Modellen verbessert.

Wir zeigen, daß die vorgestellten Konzepte für das Entwickeln von Komponentenmodellen und deren Zusammensetzung mit unterschiedlichen räumlichen und zeitlichen Skalen in einem Softwareframework zusammengeführt werden können. In anschaulichen Fallstudien verwenden wir Prototypen, mit denen wir Komponentenmodelle mit räumlichen und zeitlichen Prozeßbeschreibungen entwickeln, die unter anderem Landnutzungsänderung, hydrologische Prozesse oder Wachstum von Biomasse simulieren. Diese Komponentenmodelle nutzen wir dann um integrierte Modellen zu konstruieren.

1 Introduction

1.1 Background

The increasing demands of humans for space, natural resources and energy are the main causes for challenging environmental problems such as the loss of biodiversity, degradation of landscapes, or pollution of natural systems (e.g. United Nations, 2002; FAO, 2003). To reach the sustainable management of these limited resources it is important to assess the anthropogenic influence on the natural environment appropriately. In doing so, it is necessary to incorporate the feedback mechanisms in human–natural systems. For example, using groundwater as source for drinking water beyond the sustainable level can lead to land subsidence or even contribute to sea level rise (see Sutanudjaja, 2012). Expansion of urban areas to traditional retention areas of river systems, as an example for land use change, can result in increased flooding, and associated costs for economic damage and flood mitigation (e.g. de Kok and Grossmann, 2010).

Computer models are essential tools in the assessment of human–natural systems. The numerical representations and simulations of the environmental processes can provide insight in the potential impact of human behaviour or management decisions. Models can be used for several purposes. First, models are used to improve our scientific understanding of environmental processes (e.g. Rotmans, 1990; Claessens et al., 2009). They allow to evaluate scientific hypotheses about the structure and functioning of real–world environmental systems, and their evaluation can help to develop or improve the description of the behaviour of environmental systems (see, e.g. Jørgensen and Bendoricchio, 2001; Gurney and Nisbet, 1998; Wainwright and Mulligan, 2004; Kleinhans et al., 2010). Second, models can be used for scenario analysis. For example, scenarios for climate change and demographic development can be used as driving forces to evaluate the probable future land use (e.g. White et al., 2012). Models allow to evaluate cause–effect relationships, and can be used to support planning and management directives related to impact assessment (see, e.g. Partidário, 2000; Jakeman and Letcher, 2003; Mahmoud et al., 2009). Third, models can be used to forecast the behaviour of environmental systems. Forecasts can be used to assess hazardous emergencies threatening the health of humans, such as predicting the outcomes of catastrophic events like floods (e.g. van der Knijff et al., 2010) or forest fires (e.g. Karafyllidis and Thanailakis, 1997).

For a comprehensive assessment of human–natural systems, it is relevant to integrate knowledge from the environmental, social and economic disciplines in models. The construction of these larger models cannot be realised with the traditional mono–disciplinary modelling approaches and calls for an integrated modelling approach, i.e. combining several individual models from different disciplines to coupled systems (e.g. Argent, 2004; Hinkel, 2009; Bulatewicz et al., 2010; Laniak et al., 2013). To represent environmental systems appropriately, these integrated models need to account for the temporal dynamics on spatial entities.

1.2 Spatio–temporal modelling of systems

Numerical models can be used to simulate the changes of environmental systems over time. These are generally expressed by a state transition function that is composed of a set of equations representing the real–world processes. A model uses this transition function to calculate its current state based on its previous state, external inputs and parameters (Beck et al., 1993; Burrough, 1998).

For transient, or dynamic models respectively, a temporal discretisation of the simulation time needs to be specified. Simulation time can be treated as events (e.g. Zeigler et al., 2000), continuously or discrete. A common approach in discrete simulations is emulating the progress of a model by a repeated execution of the state transition function over a set of time steps (see, e.g. Beck et al., 1993; Burrough, 1998).

To express spatial interactions, the state transition function needs to operate on spatial entities representing geographical attributes. These entities can represent space, for example, by a set of objects as done in individual–based modelling (e.g. Grimm and Railsback, 2005). A continuous representation of spatial attributes as done in field–based modelling (e.g. Burrough, 1998) can be obtained by discretising the spatial environment into independent units such as gridded raster cells or triangular irregular networks (e.g. Tucker et al., 2001).

Building spatio–temporal models and applying these in scenario analysis and forecasting, requires modellers to perform a number of distinct steps.

1.3 Steps to build numerical models

The process of building and evaluating models to answer a specific research question is known as the model development cycle (Jørgensen and Bendoricchio, 2001; Jakeman et al., 2006). This cycle consists of different stages of conceptual development, technical model development and model evaluation, which we now will briefly outline.

The conceptual phase is aimed at identifying the structure to use for a model. This phase involves the formulation of the model objective, a clear definition of the system and its boundaries, the selection of the processes to include, the mathematical representation of these processes, and model properties such as spatial and temporal scale and available data (see, e.g. Jakeman et al., 2006; Hinkel, 2009). Result of this phase is a conceptual model that forms a starting point for the technical implementation phase.

The technical implementation phase deals with the conversion of the conceptual model into a numerical representation of a computer program. Writing the code of a model requires both the scientific understanding of processes that need to be modelled, and, preferably, understanding of basic software engineering principles to be able to design and build the environmental model efficiently and as a software product which is easy to maintain. As the development of a computer model is a complex and error prone process, model verification, i.e. “ensuring that the computer program of the computerised model and its implementation are correct” (Sargent, 2013) needs to be performed to obtain a sound numerical model (see also Oreskes et al., 1994).

The model evaluation phase follows the technical implementation phase and is relevant for the scientific integrity and credibility of the constructed numerical implementations of the model (e.g. Rykiel, 1996; Jakeman and Letcher, 2003; Jakeman et al., 2006; Letcher et al., 2006; Matott et al., 2009). The objective of the model evaluation phase is to ensure that a model "possesses a satisfactory range of accuracy consistent with the intended application of the model" (Sargent, 2013). The process representations given by model equations and parameters need to be in agreement with the observations. Initial parameter choices, for example, can produce model results that do not agree with the known behaviour of a system. It is thus required to calibrate parameter values to an optimal fit between modelled and observed values (Hill and Tiedeman, 2007), or to sequentially update state variables when observations become available (Simon, 2006).

The quantification of model uncertainties, also referred to as error propagation, assesses the model output with respect to the uncertainties and errors in the input data and model structure, and must be considered in developing any model (Jakeman et al., 2006). For complex environmental systems this is done by Monte Carlo simulation (Heuvelink, 1998) by running a large number of model realisations and computing the ensemble statistics. In case observational data are available, sequential data assimilation can be used to merge the information present in a model with uncertain data to achieve uncertainty quantification and reduction (Liu and Gupta, 2007). Applied methods here are the Ensemble Kalman filter (e.g. Weerts and El Serafy, 2006; Evensen, 2003; van Leeuwen, 2003) or the particle filter (e.g. Moradkhani et al., 2005; Doucet et al., 2000; Liu and Chen, 1998).

The model development itself evolves as an iterative, adaptive process to find an optimal model (Jakeman and Letcher, 2003). Modellers need to be able to perform the iterative development steps to understand the scientific details of modelled environmental systems.

1.4 Decomposing complex systems

In software engineering and, increasingly, environmental modelling, modularisation is a common strategy for structuring complex systems and the design process. With modularisation, complex systems are segmented into well–defined modules (see, e.g. Parnas, 1972; Kraines et al., 2005). The benefits of modularisation are in general manifold (see, e.g. Brooks, 1987; Villa and Costanza, 2000; McArthur et al., 2002): the modules have a defined scope and therefore limited complexity, enhancing the transparency and comprehensibility of the included processes. They have an encapsulated, i.e. hidden process description that can be modified, extended or exchanged without affecting the process descriptions of other modules. By restricting the functionality, the modules can be re–used in the construction of different model applications. Using modularisation therefore allows for the construction of complex systems from modules with complementary functionality.

When we look at the modular construction of integrated environmental models, what are the requirements to the development process for environmental modellers? First, a modeller must be able to perform the technical integration. That is, a modeller must

be able to specify the process descriptions and therefore the transition function for the individual component models, and finally to couple these into larger models. Second, a modeller needs to be able to assess how good a constructed model represents the modelled environmental system, for example by comparing modelled and observed data. We now briefly outline the tasks that need to be performed in the construction of an integrated model, and give examples of software packages that can provide support with these tasks.

1.4.1 Building component models

Here, a component model is defined as the main building block that modellers can use to construct integrated models. A component model is an individual, self–contained module that holds a single spatio–temporal process representation of an environmental phenomenon. To obtain reusable component models, the process implementations in the component model should be independent of a particular modelling context or other component models. The coupling of component models can be realised by means of a standardised description for the inputs and outputs.

A common approach to simulate dynamic behaviour in expressing forwarding modelling is to divide the environmental process description into the different execution phases of model initialisation, a run phase, and a finalisation phase, if necessary (see, e.g. Hill et al., 2004; Gregersen et al., 2007; Karssenberg et al., 2010; Peckham et al., 2013).

The run phase describes the progress of the component model for each time step, and it is here that the modellers implement the numerical representation of the environmental process. System programming languages such as Fortran, C or Java provide the greatest flexibility in implementing these process descriptions. However, these languages do not provide an intrinsic support needed for environmental modelling, and they require a profound knowledge for correct and optimal usage. It is preferable for environmental modellers to use operations that represent domain specific concepts, as these are more comprehensible for domain specialists and allow for a faster and less error–prone implementation of the process descriptions (Karssenberg, 2002). Software packages providing operations at the domain conceptual level are, for example, PCRaster (Wesseling et al., 1996; Karssenberg et al., 2010), SELES (Fall and Fall, 2001) or SimuMap (Pullar, 2004).

Modular development of numerical models is stimulated by using software development practices such as object–oriented programming (e.g. Booch et al., 2007) or the component–based development (e.g. Szyperski, 2002). Software frameworks using object–oriented features of system programming languages to define modules and interfaces are, for example, GEONAMICA (e.g. Hurkens et al., 2008), OMS3 (David et al., 2013), ESMF (Hill et al., 2004) or CSDMS (Peckham et al., 2013).

1.4.2 Coupling component models

Standardisation of input and output interfaces of component models and communication protocols allow for a straightforward coupling of component models into integrated models. The coupling relates output interfaces to input interfaces, and hence specifies

the runtime interaction and data exchange between component models. Prior to model execution, a modeller needs to specify which component models interact in an integrated model, what information the models will exchange, and in which direction this exchange takes place.

An ad–hoc coupling can be executed for particular application cases, using a variety of programming languages (e.g. Roberts et al., 2010) or a scripting language to glue components into an integrated model (e.g. Huang et al., 2012). Tailored software packages with an emphasis on establishing a more generic component coupling are, for example, the Typed Data Transfer library (Hinkel, 2009) or the Model Coupling Toolkit (Larson et al., 2005) realising data transfer between components, the OpenMI framework (Gregersen et al., 2007) as a communication standard for model interoperability (Moore and Tindall, 2005), or software packages with visual modelling languages such as STELLA (2013) or ExtendSim (2013).

1.4.3 Discretisation discrepancies

Modularisation allows for the construction of generic, reusable component models. Component models, however, describe individual environmental processes that operate on particular spatial and temporal discretisations. When integrated models are constructed, component models at various spatial and temporal scales need to be coupled, and discrepancies in space and time between interacting components can occur. Modifying the process descriptions and discretisations of components to resolve this issue model–specifically contradicts the principle of reusable component models. An intermediate step for scaling is therefore required when variables are exchanged. A modeller can implement this intermediate step using well–established spatial and temporal scaling methodologies (e.g. Blöschl and Sivapalan, 1995; Bierkens et al., 2000; Skøien et al., 2003).

Software tools like ExtendSim (ExtendSim, 2013) or OpenMI (e.g. Gregersen et al., 2007) provide a set of interpolation algorithms to construct scaling operations. Modelling frameworks such as the ESMF (e.g. Hill et al., 2004) or the CSDMS (e.g. Peckham et al., 2013) allow a modeller to program scaling operations at the level used in system programming languages.

1.4.4 Executing an integrated model

With component model construction and coupling, a modeller specifies the structural setup of an integrated model. To execute the constructed models, a number of issues need to be taken into consideration. First, the specified model composition needs to be sound, i.e. the exchanged variables need to comply with the spatial and temporal characteristics of the components. Second, an ordered execution of the components according to their time steps and an exchange of information at appropriate moments needs to be realised. Third, models need to be executed in different ways to support assessment features such as calibration or Monte Carlo simulations. Model execution and result generation is also related to post–processing steps such as using tools supporting the analysis of model outputs and visualisation of results.

A modeller needs to organise component execution and data exchange manually when using programming languages or gluing approaches in model construction. Software frameworks such as the HLA (e.g. Lindenschmidt et al., 2005), Tarsier (e.g. Rahman et al., 2004) or LIQUID (Branger et al., 2010) provide built–in support for the ordered execution and data exchange. Software tools supporting model analysis are, for example, PEST (Doherty and Johnston, 2003) for model calibration, or OpenDA (2013) for data assimilation.

1.4.5 The semantic integration

Component models represent scientific knowledge about particular environmental processes (McIntosh et al., 2005). The numerical implementation conducted with software tools, however, does not necessarily expose the scientific meaning of the underlying models. By using programming languages, for example, a given variable *temp* can semantically refer to a temporary variable, or to a temperature value. Deriving the meaning of variables from the syntax can be ambiguous. This raises a problem for a modeller when trying to understand and apply a component model: how can necessary information about state variables, inputs, outputs, constraints, and a valid coupling of component models be identified based on the numerical implementation? In addition to a technical interface, a semantic interface describing the functioning of a component model on a scientific level is desired.

Common vocabularies shared between different disciplines can help to improve the scientific interoperability of component models. These vocabularies can be developed by standardising the semantics of component models and their relationships by using, for example, metadata descriptions (e.g. Nogueras-Iso et al., 2004) or ontologies (Uschold and Gruninger, 1996). Such standardised approaches are used, for example, in describing individual environmental attributes (e.g. WaterML, 2013; NetCDF-CF, 2013), or integrating geographic information (e.g. Buccella et al., 2009; Couclelis, 2010) or application cases that model human–natural systems (e.g. Polhill and Gotts, 2009; Beck et al., 2010).

1.5 Research questions and thesis outline

The application of an integrated model, i.e. improving scientific understanding, scenario analysis and forecasting, falls within the scope of domain specialists. They need to be able to perform the technical steps, i.e. the construction of component models, their technical and semantical integration into coupled models, and the analysis thereof. Using an ad–hoc conglomeration of the abovementioned software tools, however, hampers domain specialists such as hydrologists in a flexible construction and analysis process.

The general objective of this dissertation is to provide a software framework for domain specialists to construct and analyse multi–scale integrated models within a single modelling platform. To reach this objective, the following research questions are examined in the individual chapters of this dissertation:

1. *How can multi–scale spatio–temporal integrated models be executed, given that their construction follows a component–based development process?*
   For a sound execution of integrated models, an appropriate order of execution and data exchange needs to be determined based on the time steps of the individual component models and the couplings specified by a modeller. The first two chapters of this dissertation use two different approaches for the execution of such integrated models. Firstly, we use a client–server approach with a central instance organising a shared time line between component models allowing to derive a schedule for component execution (Chapter 2). Secondly, we use an execution scheme determined by a request–reply based specification of the coupling of component models, and explore its application in uncertainty analysis (Chapter 3).

2. *How can generic building blocks bridge spatial and temporal discretisation differences?*
   Component–based development of environmental component models implies that models can act as individual entities with confined process descriptions and hold their own spatial and temporal discretisations. Therefore, these component models can be used as reusable building blocks in the construction of integrated models. At coupling, however, differences in discretisations may appear and need to be bridged. Starting from the assumption that the implementations of the component models should not be modified to avoid reducing their generic application, Chapter 4 explores a programmable building block to bridge the spatial and temporal discretisation differences.

3. *How can modellers integrated legacy applications into coupled models?*
   The component models constructed in the Chapters 2 and 3 were built within the modelling environment. A common use case in the construction of integrated models, however, is the integration of functionalities provided by existing legacy applications. We explore two different approaches to achieve the integration of legacy applications into component models. First, we explore the wrapper approach in a coupling of an existing application modelling land use change to a hydrological model (Chapter 4). Second, we investigate how the technical means of an existing modelling environment can be used to couple legacy applications (Chapter 5).

4. *What is the relevant information needed to couple component models independently of their implementation?*
   The Chapters 2 to 5 describe the different technical approaches for the integration of interdisciplinary component models. Still, next to the numerical implementation component models hold a scientific meaning. Chapter 6 shows how component models and integrated models can be described based on scientific meaning and how a coupling can be evaluated based on this meaning and independently of technical implementation used.

The synthesis in Chapter 7 summarises answers to the research questions, provides main conclusions and outlines new questions for future research in the development of integrated models.

2 Algebras for integrated model building

This chapter is based on:
SCHMITZ, O., KARSSENBERG, D., DE JONG, K., DE KOK, J.-L. AND DE JONG, S. M. (2013), Map algebra and model algebra for integrated model building. Environmental Modelling & Software 48, 113–128.

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.

2.1 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 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

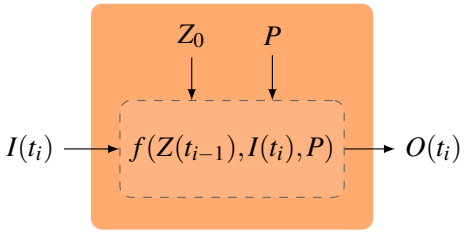

Figure 2.1 Conceptualisation of a model component. The external interface describes the required input variables I and provided output variables O per time step t_i. The implementation of the modelled process is internal. Starting from an initial state Z_0, the transition function f transfers the component state Z into a new state. P is a set of parameters.

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 2.3, we illustrate the coupling of model components with different discretisations by means of accumulators. We introduce Python as a modelling environment in Section 2.4 and discuss the technical design enabling the ordered execution of components and accumulators. Section 2.5 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 chapter.

2.2 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.

2.2.1 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. Figure 2.1 shows the conceptual separation of a model component into internal and external functionality. This separation avoids a monolithic setup of integrated models

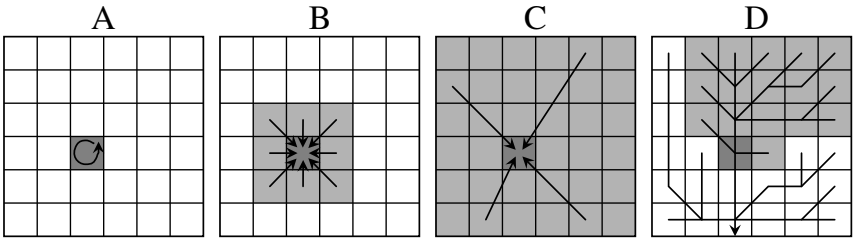

Figure 2.2 Classification of map algebra operations. (A) Point or local, (B) direct neighbourhood or focal, (C) entire neighbourhood or zonal, and (D) operations on a neighbourhood given by a specific topology (Karssenberg and de Jong, 2005a).

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):

$$Z(t_i) = f(Z(t_{i-1}), I(t_i), P) \quad \forall\ t_i \tag{2.1}$$

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 2.1. A 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, 2005a) (Figure 2.2):

(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 2.1 (c.f. Wesseling et al., 1996; Pullar, 2001, 2003; Cerveira Cordeiro et al., 2009; Mennis, 2010).

2.2.2 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 2.1 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:

$$I_l(t_i) = A(E_n(t_i), O_n(t_{[h,i-1]})) \tag{2.2}$$

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

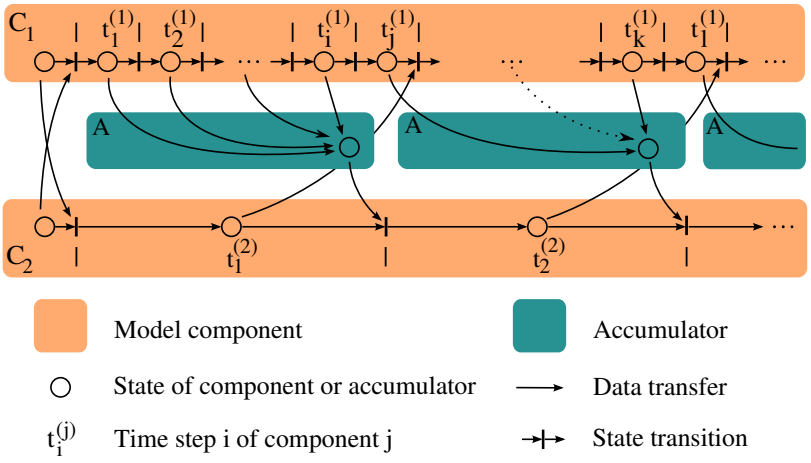

Figure 2.3 Temporal control flow between interacting model components with shorter (C_1) and longer (C_2) time steps. Each model component requests the most recent output from the other component. While C_1 directly accepts the input of C_2, C_2 expects aggregated values from C_1, provided by the accumulator A. Note that data conversion issues such as spatial resampling are not explicitly included in this figure.

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.

2.3 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.

2.3.1 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.

Figure 2.3 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 latest value (i.e. ${t_i}^{(1)}$ or ${t_k}^{(1)}$ in Figure 2.3) 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.

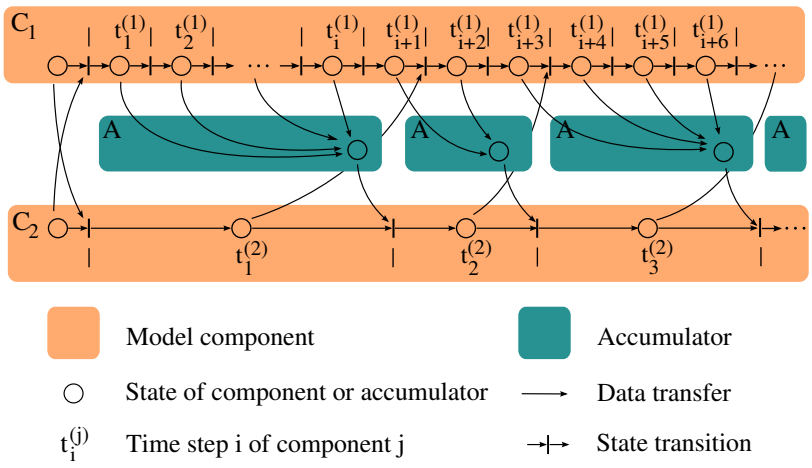

Figure 2.4 Temporal control flow between two interacting components where model component C_2 changes the time step duration. The scheduler needs to update the interval considered by the accumulator and to notify C_1 of the advanced availability of output variables from C_2.

2.3.2 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.

Figure 2.4 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.

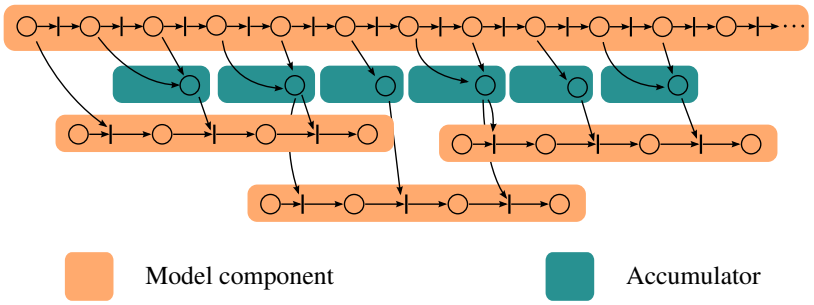

Figure 2.5 Components with limited life time interact with a component with continuous time steps. The scheduling needs to update the interval considered by the accumulator and to keep track of the number of individual components. Also, aggregated values are used as initial values for the individual components.

This alternating scheme of component execution and schedule generation continues until the end of the simulation.

2.3.3 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 Figure 2.5) as in the case of agent–based models. An example is an integrated model where a field–based groundwater component is coupled to a 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

Table 2.1 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

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.

2.3.4 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

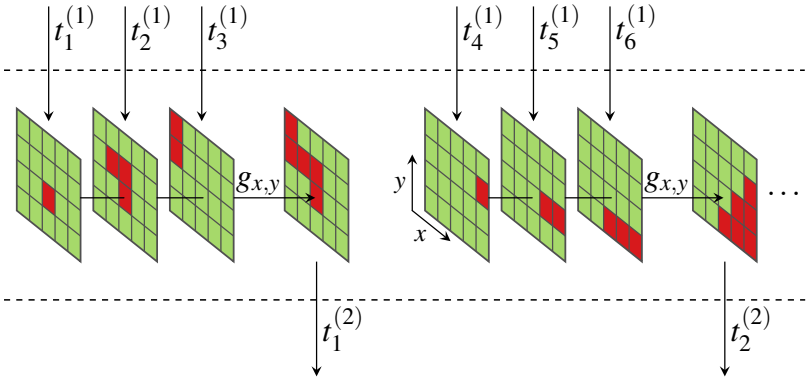

Figure 2.6 Schematic workflow of an accumulator connecting two components with different time steps $t_i^{(1)}$ and $t_j^{(2)}$. The operation g, here the Boolean `OR`, aggregates the set of input maps obtained since the last exchange moment. Cell values are coloured in red for `True` and green for `False`. Left to right: time; x and y represent spatial coordinates.

for 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 2.1. These generic accumulators provide point operations executed on a temporal interval. Figure 2.6 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.

2.4 Technical implementation

Following the concepts described in the previous section, we developed a prototype framework consisting of two layers (Figure 2.7). The layer for custom model development provides the modeller with the instruments to describe and design model components and coupled models. A second layer provides the execution management for schedule generation, model execution and data transfer. The concepts introduced in this section

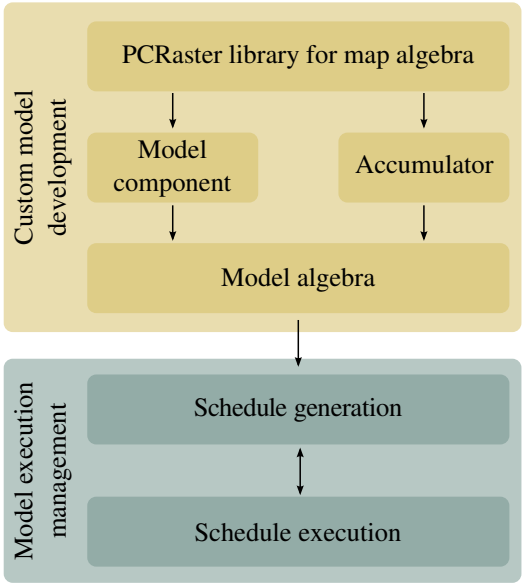

Figure 2.7 Architecture of the modelling framework. The custom model development layer provides framework functionality for component and model building. The model execution layer comprises of the schedule generation and execution management.

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.

2.4.1 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.

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. XML, 2008; HDF5, 2010; GDAL Development Team, 2013; OGR, 2013), and analysis or visualisation tools (e.g. Schroeder et al., 2000; Hunter, 2007; RPy, 2011) 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 (Van Der Walt et al., 2011; SciPy, 2013) 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 (2013), 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.

2.4.2 Custom model development

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, 2005a; 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, 2013). 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 2.1. For instance, the `windowtotal` operation from the PCRaster module implements a direct neighbourhood operation (Figure 2.2B):

```
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.

Framework implementation

Within the layer for the custom model development (Figure 2.7), the framework provides templates to allow for the development of model components and the use of accumulators, and their coupling to integrated models. Figure 2.8 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.

The diagram of Figure 2.8A 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

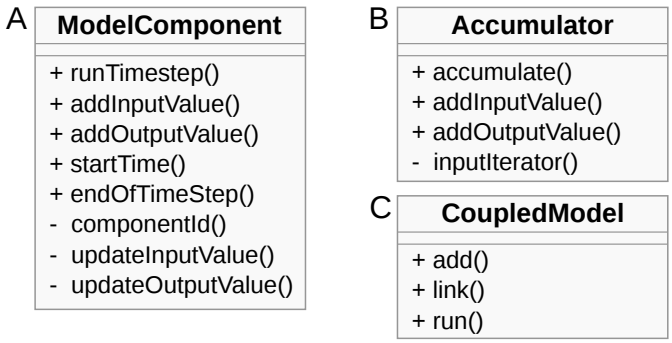

Figure 2.8 UML diagram of the classes provided by the framework for model development. ModelComponent (A) needs to be filled with functionality by the model builder. Accumulator (B) and CoupledModel (C) provide methods to specify the integrated model.

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 Figure 2.8B 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.

Figure 2.8C 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.

2.4.3 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 2.2 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

Table 2.2 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
```

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.

2.5 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.

2.5.1 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 27925 cells with a cell length of 10 m.

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 com-

Table 2.3 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")
```

ponent 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 2.3 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. Figure 2.9 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.

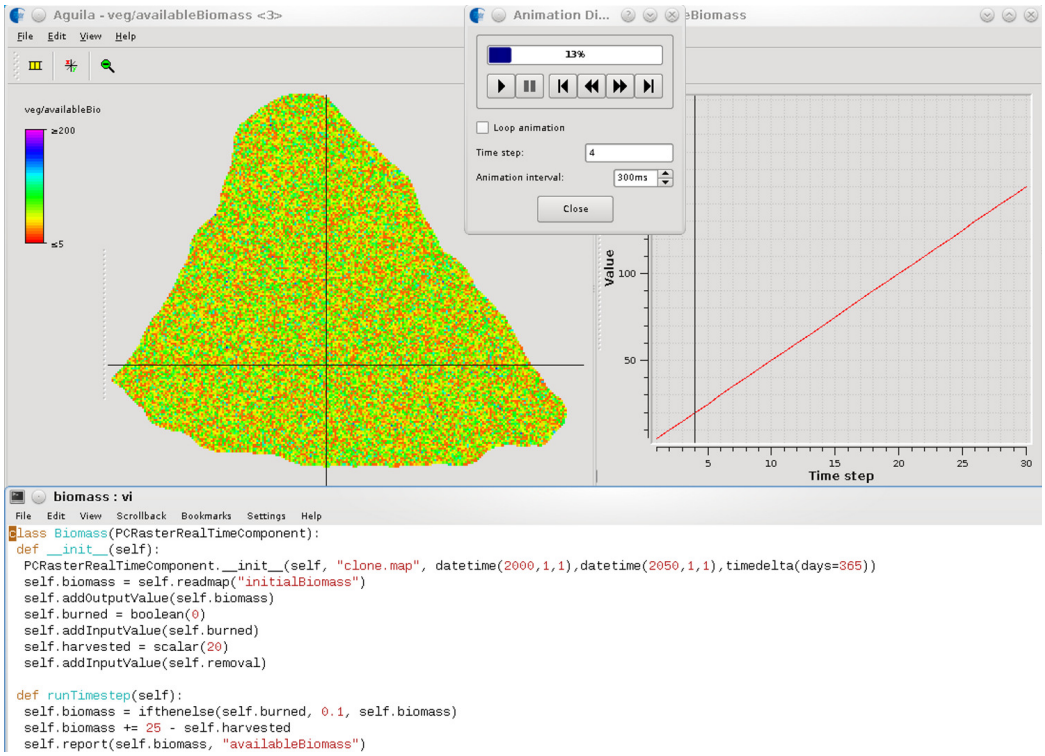

Figure 2.9 Screen capture showing the spatial distribution of biomass and the timeseries for a specific cell. Cell locations and instants of time can be selected interactively.

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 2.4. 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`

Table 2.4 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() > 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))
```

Table 2.5 Model algebra script specifying the coupled model consisting of the two model components, a logical accumulator, and their interactions.

```
1 biomassFire = CoupledModel()
2 biomass = Biomass()
3 fire = Fire()
4
5 biomassFire += biomass
6 biomassFire += fire
7
8 biomassFire.link(biomass["biomass"], fire["availableBiomass"])
9 biomassFire.link(fire["burnedArea"], biomass["burned"], OR)
10
11 scheduler.SingleRun(biomassFire).run()
```

(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 hours (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).

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 Figure 2.2. The resulting output variable holds the total burned area of the current year.

Component coupling and results

The model algebra script that is used by the modeller to implement the coupled model is shown in Table 2.5. 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`

Table 2.6 Python script for the improved biomass component. The input and output interface to other components remains the same, only the process description is modified.

```
11  def runTimestep(self):
12    self.biomass = ifthenelse(self.burned, 0.1, self.biomass)
13    growth = 0.2 * self.biomass * (1 - self.biomass / 100.0) - ↩
          (self.biomass**2 / (self.biomass**2 + 1))
14    sumBiomassOfNeighbours = window4total(self.biomass)
15    numberOfNeighbours = window4total(spatial(scalar(1)))
16    diffusion = 0.01 * (sumBiomassOfNeighbours - numberOfNeighbours * ↩
          self.biomass)
17    growth = growth + diffusion
18    self.biomass = self.biomass + growth
19    self.report(self.biomass, "availableBiomass")
```

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.

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 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):

$$\frac{dB}{dt} = rB\left(1-\frac{B}{k}\right) - \left(c_0 + c_{inc}t\right)\frac{B^2}{B^2+1} + dD_{x,y} + \epsilon \tag{2.3}$$

The script in Table 2.6 shows the more sophisticated biomass component, where the first ten lines are the same as in Table 2.3. Equation 2.3 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).

Figure 2.10 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

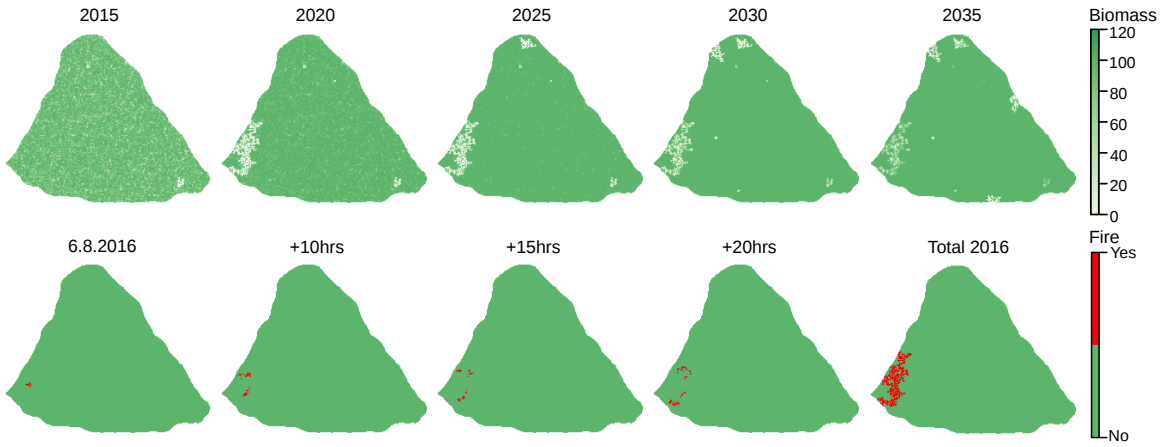

Figure 2.10 Result maps from a model run with the improved biomass component. The top row shows biomass values, the bottom row output of the fire component and the accumulator.

incident occurring in August 2016. The bottom right map shows the total burned area accumulated in the year 2016.

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 2.3 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 2.7. 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 Figure 2.11). 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 2.5, 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` components. Figure 2.11 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 (Figure 2.11). Also, the fire in the south–west of the catchment reduces the area of biomass available for logging (Figure 2.11B). Figure 2.11C 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 (Figure 2.11D).

Table 2.7 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")
```

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; Chapter 5). 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 2.4.1 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

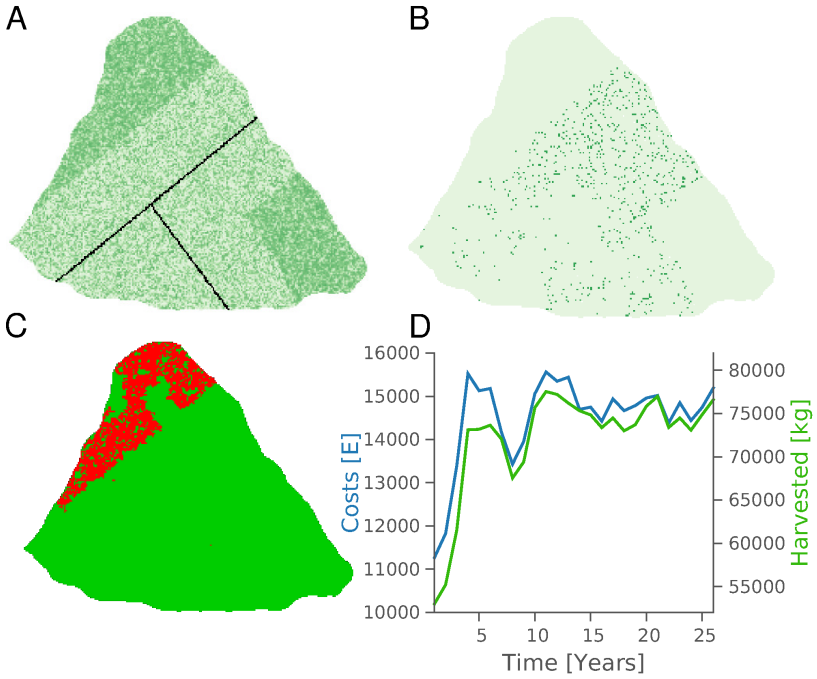

Figure 2.11 Outputs of the integrated model extended by the harvest model component. (A) Growth variable from the `Biomass` component with overlayed road infrastructure. (B) Reduced harvesting area in the `Logging` component due to a fire incident. (C) Accumulated output of the `Fire` component. (D) Trend of total costs and harvested biomass.

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. Saiful Islam and Piasecki, 2006; Madin et al., 2007), 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 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.

2.6 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. Karssenberg et al., 2010; Van Der Walt et al., 2011; Kraft et al., 2011; ArcGIS, 2013; SciPy, 2013; Oliphant, 2007). 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.

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; Chapter 5). 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).

3 Request–reply execution of coupled multi–scale component models

This chapter is based on:
SCHMITZ, O., DE KOK, J.-L., DE JONG, K. AND KARSSENBERG, D.
Request–reply execution of coupled multi–scale component models. In preparation.

Abstract

Dynamic environmental models use a state transition function, external inputs and parameters to simulate the change of real–world processes over time. Modellers construct the state transition function and specify the external inputs that are required in the process calculation of each time step. Depending on the usage of a component model such as standalone execution or in an integrated model, the source of the external input needs to be selected. The required external inputs can thereby be obtained by a file operation, in case of a standalone execution; or inputs can be obtained from other component models, when the component model is used in an integrated model. Using different notations to specify these input requirements, however, requires a modification of the state transition function per application case and therefore would reduce the generic nature of a component model.

To address this, we propose in this chapter the function object notation as a means to specify the input requirements of a component model. The function object notation provides modellers with a uniform syntax to express input requirements within the state transition function. At component initialisation, the function objects can be parametrised with different external sources. In addition to a uniform syntax, the function object notation allows a modeller to specify a request–reply execution flow of the coupled models. We extend the request–reply execution approach to Monte Carlo simulations and implement a software framework prototype. Using this prototype, we build an exemplary integrated model by coupling components that model land use change, hydrology and eucalyptus tree growth at different temporal discretisations to obtain the probability for bioenergy plantations in a hypothetical catchment. The presented approach allows modellers to specify input requirements in the state transition function independently from the source of external inputs and therefore increases the reusability of component models.

3.1 Introduction

Integrated environmental research faces the practical challenge of combining knowledge across different scientific disciplines such as hydrology, ecology, or crop science.

Spatio–temporal models from these various disciplines may operate at different spatial and temporal discretisations, depending on the processes represented. Coupling such models allows for analysis from an integrated perspective, improving our understanding of interactions between subsystems (e.g. Rotmans, 1990; Verburg, 2006; Claessens et al., 2009; Elag et al., 2011). Moreover, such integrated models are an important tool for environmental management and integrated assessment (e.g. van Ittersum et al., 2008; Ewert et al., 2009; Muth Jr. and Bryden, 2013; Welsh et al., 2013). To obtain reliable results from these interdisciplinary models and to appropriately represent complex environmental systems, modellers are faced with two major challenges: (1) the construction of component models and their couplings such that an appropriate representation is given of the specific system under study, and (2) error analysis to assess the reliability of their models and the uncertainty in particular forecasts required for managing the real systems.

Model builders can choose from a variety of software applications that support environmental modelling, including, amongst others, general purpose programming languages (e.g. Fortran, C, Python), geographical information systems (e.g. ArcGIS, 2013; Quantum GIS Development Team, 2013), visual and domain specific modelling languages (e.g. STELLA, 2013; ExtendSim, 2013; PCRaster, 2013), and software frameworks tailored to the development of integrated models (e.g. CSDMS, 2013; OpenMI, 2013; TDT, 2010). Software development practices emphasising a modular development of reusable components are, for example, object–oriented development or component–based software engineering practices (e.g. Szyperski, 2002; Booch et al., 2007). Further existing software applications and software engineering practices are given in the general introduction in Chapter 1, in Section 2.1 and in Section 4.2.1.

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

In summary, modellers need to construct and couple component models to integrated systems, and modellers need to analyse the constructed models. This is done in an iterative development process (e.g. Jørgensen and Bendoricchio, 2001; Jakeman et al., 2006) by using different process representations and different component compositions to test and evaluate underlying assumptions. Flexibility is required to be able to perform such a plug–and–play construction and analysis (see, e.g. Papajorgji, 2005; De Kok et al.,

2010). Modular development practices such as the component–based development practice (e.g. Szyperski, 2002) promote this flexibility by providing statements of requirements relating to the 1) the construction of confined components with defined input and output interfaces and 2) the provision of a communication protocol that is shared between all components. Modellers should be preferably able to follow these practices within one modelling environment on their level of expertise (see also Karssenberg, 2002).

At component construction, the modeller specifies the input requirements of a component model. Depending on the usage of a component model, however, these requirements can be met at model runtime differently: if a component model is executed individually, the inputs can be obtained from precalculated data or observational data stored on disk. If a component model is part of an integrated model, the inputs are obtained from other component models that calculate data at runtime. The software prototype in Chapter 2 uses different functions to obtain inputs either from disk or from other component models. The component interfaces therefore depend on how input variables are obtained. This adds a dependency between the component interface and the process description. The resulting component models are less generic, although obtaining input variables from disk and from other component models is conceptually equal. It is required to disband the dependency between process description and component interface to enhance the reusability of component models.

The objective of this chapter is to develop a mechanism allowing model builders to uniformly describe process descriptions independently of the origin of component inputs. To reach this objective, we first generalise the input requirements of a component model and develop in Section 3.2 a parametrisation of the component interface allowing for different input sources. We propose the function notation as consistent syntax to describe input origins in the process descriptions. Based on that notation we introduce the concept of function–based model execution, and outline the requirements for the execution of multi–scale spatio–temporal component models. Then, we extend the requirements to allow for uncertainty assessment based on Monte Carlo simulation. Based on the requirements, we describe in Section 3.3 a framework prototype developed in the high–level language Python (2013). A case study in Section 3.4 illustrates the application of the modelling framework for the construction, execution and visualisation of stochastic spatio–temporal component models. We conclude this chapter with an evaluation of the presented approach.

3.2 Methodology

3.2.1 Specification of data requests

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

$$Z(t_i) = f(Z(t_{i-1}), I(t_i), P) \quad \forall\ t_i \tag{3.1}$$

The state transition function f is in the numerical model represented by a set of predefined operations that allow for a flexible construction process. These operations are calculated on spatial data types, provided for example by the map algebra concept and related implementations (e.g. Tomlin, 1990; Karssenberg et al., 2007; Chapter 2). The kind of external inputs $I(t_i)$ need to be specified by the modellers as well. These inputs can be obtained, for example, by an operation reading a data set from disk, or they can be provided as output from another component model. The application of the component model can necessitate obtaining inputs from different data sources. When executing a component model individually, required inputs need to be obtained by reading data sets from disk storage. When executing an integrated model, required inputs can be provided by and obtained from other component models.

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

```
head = read("levels")
```

with `read` an operation reading a data set from disk holding groundwater head values in a catchment. Specifying process implementations independent of spatial and temporal discretisations, such as the catchment extent and spatial discretisation or the time step, results in a more generic component. A component is therefore easier to reuse with different catchments or simulation periods. However, using a `read` operation in the state transition function fixes the assignment of the head values to a read operation from disk. A modeller therefore needs to modify the process implementation in case that other sources of the required inputs, for instance another component model that is run simultaneously, are available.

To allow for a generic process description independent from an input either obtained as data read from hard disk, or calculated by another component model, a consistent syntax is required to express input requirements in the process descriptions. To provide such a syntax, we can apply the function objects concept known from programming languages such as C++, Java or Python. Function objects provide a syntax allowing to call an object, equivalent to a component model in the field of environmental modelling, with the common syntax of a function call. A modeller can therefore specify an input requirement within the transition function f as function call:

```
head = groundwater("levels")
```

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

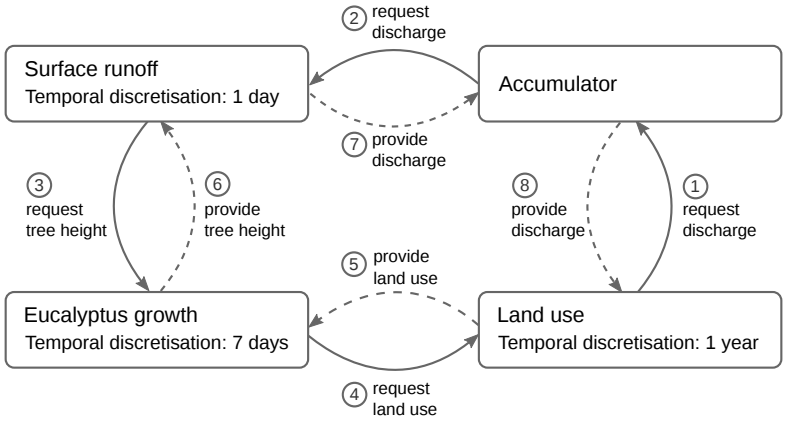

Figure 3.1 Conceptual setup of an integrated model simulating the spatial allocation of bioenergy crops. The numbers indicate the order of steps performed in request–reply approach. The land use change model initiates the first input request.

3.2.2 Execution of data requests

The function notation given above provides modellers with a consistent syntax expressing the origin of data and therefore the data exchange: either as an intrinsic operation of a modelling environment, or expressing a link to another component in a coupled model. Next to the specification of the data exchange, the function syntax provides a means to specify the execution flow of component models (Bulatewicz et al., 2013). A component model requests a certain input variable, waits until the variable has been retrieved, and continues with its process calculation. In multi–scale models with component models having different temporal discretisations, a request–reply based function call requires to incorporate time for a correct execution of integrated models.

To illustrate the concept of request–reply based model execution, we consider an integrated model simulating a system that regulates the spatial allocation of bioenergy crops, for instance eucalyptus. The representation of the system is simplified, as the purpose of the model is mainly to illustrate the concepts. The integrated model includes three component models: a land use change model, a hydrological model, and a tree growth model (see Figure 3.1). The land use change model spatially allocates expanding eucalyptus, using a time step of a year. Expansion of eucalyptus will reduce stream discharge, as the water use of eucalyptus is high. Thus, it is assumed that governmental laws are in place that each year restrict eucalyptus expansion to catchments with a stream discharge in the previous year that was still above a threshold value, i.e. the required environmental flow. To model this, the land use change model requires yearly discharge for all catchments for each year. A hydrological model running at a time step of one day is used to calculate the discharge. The land use change model, running at a time step of a year, requests the yearly discharge from an accumulating model component, which converts daily stream discharge requested from the hydrological model to yearly values. As stream discharge depends on evapotranspiration, which is assumed here to be a function of the tree height, the hydrological model requests the tree height from a tree

growth model, which is the third model component. The tree growth model simulates the growth of newly allocated eucalyptus trees, using a time step of a week. The model requests yearly updates of the map with the simulated eucalyptus plantations from the land use change model.

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

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

3.2.3 Support for uncertainty analysis

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

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

This procedure needs to be extended to all components in coupled models, thereby considering that each component model can have its own spatial and temporal discreti-

Table 3.1 Monte Carlo scheme for the request–reply execution of coupled component models.

```
Step 1:
   for all c in C:
     generate a set S with parameters and inputs suitable for component c

Step 2:
   run_until(ci, tj):
     if input i from ck required:
       if not ck at tj:
         run_until(ck, tj)
       obtain input i
     while ti < tj:
       calculate state transition function

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

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

sation. For a set of $C = (1, ..., N)$ component models in an integrated model, the Monte Carlo simulation is executed as described in Table 3.1.

Following the procedure above, the stochastic parameters and the stochastic input variables for all component models are generated first (step 1). To perform a request–based execution of all realisations (step 2), we define a recursive function `run_until` that calculates the state transition function of a component model until a requested time step. The `run_until` function initiates a subsequent execution of a coupled component model to obtain an external input, if necessary. All realisations can now be calculated by executing the `run_until` function for each time step of the first component model. Finally, for each component model the ensemble statistics are calculated (step 3).

3.3 The prototype implementation of the modelling framework

An objective of the modelling framework prototype is to provide model builders with one practical environment for the construction and analysis of integrated models. We assume that environmental scientists without explicit training in software development are potential users of the modelling framework. Therefore, we provide a prototype implementation in Python (2013), allowing scientists to apply the framework after a short period of familiarisation with the scripting language. The modelling framework provides

Table 3.2 Python template showing the template of a component model and the usage of the modelling frameworks.

```
class Model(DynamicModel, MonteCarloModel):
  def __init__(self, start, end, delta, cloneMap):
    DynamicModel.__init__(self, start, end, delta)
    MonteCarloModel.__init__(self)
    self.weather = Input(WeatherModel(start, end, timedelta(days=1)))
    self.landuse = Input("Landuse")

  def premcloop(self):
    self.generateRealisations(...)

  def initial(self):
    self.state = spatial(boolean(0))
    self.export(self.state, "state")

  def dynamic(self):
    curr_weather = self.weather(self.current_time_step())
    curr_lu = self.landuse(self.current_time_step())
    recharge = curr_weather.precipitation - curr_weather.evapotrans ...
    self.state = ...

  def postmcloop(self):
    self.percentiles(...)

model = Model(datetime(2000, 1, 1), datetime(2010, 1, 1), ↩
    timedelta(days=1), "clone.map")
dynModel = DynamicFramework(model)
mcFrw = MonteCarloFramework(dynModel, nrRealisations=50).run()
```

a set of Python classes that support the construction of coupled spatio–temporal models. The approach presented here is comparable to the one presented by Karssenberg et al. (2010). Operations from the PCRaster package (2013) can be used to include spatial operations and for spatially explicit visualisations of model results.

3.3.1 Framework classes to build a component model

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

The DynamicModel class

The modeller uses Python classes as a basis for a component model, and can add support for dynamic modelling by deriving from the `DynamicModel` base class. A template showing the general structure of a component model is given in Table 3.2. The `init` section initialises the `Model` class as a component model with a simulation period and time step, and initialises the functionality for executing Monte Carlo simulations, if necessary (lines 3 and 4). The `Input` class is a supporting construct that allows a modeller to specify the origin of an input variable. The class returns a function object that can be used to express input requests within the process description given in the `dynamic` section. A modeller can use the `Input` class either to create a new instance of a coupled component model such as for the `WeatherModel` (line 5), or to refer to a previously instantiated component model such as the `Landuse` model (line 6).

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

A modeller implements the operations that are executed for every time step in the `dynamic` section. Here, data requests to an external component model can be specified (lines 16 and 17). The current time step as argument is used by the external component to update its state, if necessary. The function call returns an object holding all state variables of the queried component model. A modeller can use this object in the remainder of the `dynamic` section to access the values of specific state variables by using the dot notation such as the precipitation and evapotranspiration values from the weather model (line 18).

The MonteCarloModel class

The `MonteCarloModel` base class provides a modeller with the required functionality to execute a model for a given number of realisations. The class requires a modeller to implement the `premcloop` (Table 3.2, line 8) and `postmcloop` (line 21) methods. The `premcloop` method is executed once for each component model at the start of the simulation. The modeller can use this method, for example, to calculate variables that are constant for all realisations. After all realisations are calculated, the `postmcloop` method is executed for each component model. Here, modellers can insert operations calculating ensemble statistics from the obtained probability distribution functions, such as variance or quantiles of the stochastic model variables (see, e.g. Karssenberg et al., 2010).

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

Technical access to external component models

The modelling framework allows modellers to express a request–reply mechanism between two interacting component models. To realise this mechanism, the modelling framework needs references to class instances of external component models at model runtime. These references are required to access component models and to realise the data exchange. These references can directly be created by a modeller when instan-

tiating a component model such as is done for the `WeatherModel` in the component model shown in Table 3.2 (line 5). The `Input` class can obtain a reference to the new class instance, and the modelling framework can use this reference to realise component interactions.

Component models, however, do not always obtain direct references to class instances of external component models that need to provide data. Such a case is given in the bioenergy crop allocation example (Figure 3.1). First, the land use component requires data from the hydrological model, and the modeller specifies within the land use component a reference to the total runoff component, and therein a reference to the hydrological component model. The framework can use these given references to access the corresponding component models at the moment of the data request. The tree growth model is the last component model in the initialisation chain and only referenced from the hydrological model. The tree growth model requiring land use as input, however, has no direct reference to the land use component. In this case, the modelling framework cannot directly resolve a reference to the land use model and is thus not able to perform a data request.

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

3.3.2 Framework classes to execute component models

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

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

the main component. After the initialisation of all components in the model, the main component executes its `dynamic` method, and with it subsequently the `dynamic` sections of the coupled components.

The `MonteCarloFramework` execution framework provides the sequential execution of a dynamic model for a given number of independent realisations. The framework takes an instance of the `DynamicFramework` as first argument, and hence an instance of a component model or an integrated model. The second argument to the framework is the number N of realisations to be executed. The framework executes the scheme given in Table 3.1 with the `run` method. First, the framework executes the `premcloop` methods of all component models, and then executes N realisations (50 realisations in Table 3.2). Finally, the `postmcloop` methods of all component models are called.

3.4 Case study

We now demonstrate the concepts and functionality of the modelling framework prototype by implementing an integrated model. We use the component models and interactions of the bioenergy crop allocation example introduced in Section 3.2 (Figure 3.1). Each of the domain models, i.e. land use change, hydrology and vegetation growth is represented by an individual framework component class. The land use change, tree growth and hydrology components run with yearly, weekly and daily time steps, respectively. The aggregation of the surface runoff is required before it can be used as input to the land use change component, and is thus represented by a fourth component which does solely aggregation, as shown in Figure 3.1.

We use a hypothetical 1098 km^2 catchment with elevation between 100 and 1896 m and a cell length of 750 m (Figure 3.2A). The simulation period was set from the years 2010 to 2100. The land use change and the hydrological processes are represented by stochastic component models. We therefore run a Monte Carlo simulation using 500 realisations to obtain the probability for eucalyptus tree growing at a particular location in the catchment.

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

3.4.1 Domain models

Land use change component

The model script for the land use change component is shown in Table 3.3. The land use change component needs yearly–accumulated runoff values to calculate its time step. In the `init` section, an instance of the `TotalRunoff` class is created with a daily time step, allowing to obtain the corresponding output variable (line 6).

Table 3.3 Model script showing the land use change component and the modelling frameworks used.

```
class Landuse(DynamicModel, MonteCarloModel):
  def __init__(self, start, end, delta, cloneMap):
    DynamicModel.__init__(self, start, end, delta)
    MonteCarloModel.__init__(self)
    setclone(cloneMap)
    self.tot_runoff = Input(TotalRunoff(start, end, timedelta(days=1), ↩
        cloneMap))

  def premcloop(self):
    self.yearlyRunoffRequired = self.readmap("requiredRunoff")
    self.mainStreams = self.readmap("streams")
    self.ldd = lddcreate("dem.map", 1e31, 1e31, 1e31, 1e31)

  def initial(self):
    self.trees = spatial(boolean(0))
    self.export(self.trees, "trees")

  def dynamic(self):
    total_runoff = self.tot_runoff(self.current_time_step())
    runoffTooLow = self.mainStreams & (total_runoff.totRunoff < ↩
        self.yearlyRunoffRequired)
    noNewTrees = catchment(self.ldd, runoffTooLow)
    suitabilityNeighbourhood = ifthenelse(window4total(scalar(self.trees)) ↩
        > 0.5, scalar(1), 0)
    suitabilityRandom = ifthen(pcrnot(self.trees), uniform(1))
    suitability = ifthen(pcrnot(noNewTrees), (suitabilityNeighbourhood + ↩
        suitabilityRandom) / 2.0)
    suitSort = order(0.0 - suitability)
    self.trees = pcror(self.trees, cover(suitSort < 11, 0))
    self.export(self.trees, "trees")

  def postmcloop(self):
    mc_probability("trees", self.sampleNumbers(), self.timeSteps())

model = Landuse(datetime(2010, 1, 1), datetime(2100, 1, 1), ↩
    timedelta(days=365), "clone.map")
dynModel = DynamicFramework(model)
mcFrw = MonteCarloFramework(dynModel, nrRealisations=500).run()
```

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

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

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

The `postmcloop` section is executed after all realisations were executed. Here, a modeller can insert the operations calculating ensemble statistics, for example to calculate the probability of eucalyptus planting in each cell with the `mc_probability` operation (Table 3.3 line 29). This operation iterates over each time step and each sample, and returns probability maps for each time step (see Figure 3.2).

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

Table 3.4 Model script for the total runoff accumulation.

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

Runoff accumulation

Table 3.4 shows the implementation to calculate yearly total runoff values. The component is initialised from the land use change component (see Table 3.3, line 6) with a daily time step and thus matching the hydrological component model. In the `init` section of the component a link to the component providing surface runoff is established (line 6). For each time step, the variable holding surface runoff is obtained component and added (lines 16–17). The `reset` method is executed when the land use change component initiates the input request and before the `TotalRunoff` component executes its 365 time steps to proceed until the requested time step of the land use change component. Then, the variable holding the total runoff values is reset to zero (line 9), enabling `TotalRunoff` to act as an accumulator aggregating over yearly intervals.

Hydrological model

The script for the component modelling the hydrological processes is shown in Table 3.5. Surface runoff depends on evapotranspiration and therefore the vegetation in the catchment, thus a link to the component providing the tree height is established in the `init` method (line 6). In the `initial` section, a map with the local drain directions (Burrough, 1998) is calculated describing the flow network in the catchment. For each time step the vegetation heights are obtained by a request to the tree growth model, and used to calculate the proportion of rainfall that reaches the soil (lines 14–15), using a simple linear relation applied here for reasons of brevity. The daily precipitation is assumed as a stochastic input `rain` $\sim N(0.002, 1)$ (line 16), which is then multiplied by the proportion

Table 3.5 Model script to calculate the surface runoff.

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

of rainfall `netRainProp` to determine the generated amount of water reaching the soil surface. This amount is accumulated over the flow network using the `accuflux` operation to obtain the discharge for each day (lines 17–18).

Tree growth model

Table 3.6 shows the implementation of the component modelling the vegetation growth of the bioenergy plants. The vegetation growth component is initialised from the runoff component with a weekly time step. We assume a constant growth of the eucalyptus trees by 2 cm per week (line 12), and a complete removal of biomass in case that the trees exceed a height of 10 m (line 13). Areas that are used for planting of bioenergy crops are calculated in the land use change component, and the latest state is obtained in line 14. This information is used to determine the areas that are not occupied by eucalyptus trees, where we assume a constant vegetation height of 0.2 m (line 15).

Model output results

The model outputs and ensemble statistics can be explored interactively using the Aguila visualisation tool (Pebesma et al., 2007). Figure 3.2A shows the topography and flow direction network of the catchment, and the main streams where discharge is compared with threshold discharge, i.e. the required environmental flow. The threshold discharge is assumed to be for each cell a fixed proportion of the discharge for the situation without bioenergy. Note that the streams shown in blue increase in size from right to left, due to

Table 3.6 Model script for the eucalyptus tree growth component.

```
class Vegetation(DynamicModel, MonteCarloModel):
  def __init__(self, start, end, delta, cloneMap):
    DynamicModel.__init__(self, start, end, delta)
    MonteCarloModel.__init__(self)
    setclone(cloneMap)
    self.landuse = Input("Landuse")

  def initial(self):
    self.vegHeight = scalar(0)
    self.export(self.vegHeight, "vegHeight")

  def dynamic(self):
    self.vegHeight = self.vegHeight + 0.02
    self.vegHeight = ifthenelse(self.vegHeight >= 10.0, 0.0, self.vegHeight)
    landuse = self.landuse(self.current_time_step())
    self.vegHeight = ifthenelse(landuse.trees, self.vegHeight, 0.2)
    self.export(self.vegHeight, "vegHeight")
```

an increase in catchment area from right to left. Figure 3.2B, Figure 3.2C and Figure 3.2D show exemplary ensemble results calculated from 500 realisations of the Monte Carlo simulation. Figure 3.2B shows, for the year 2045, the probability that a cell is excluded from bioenergy allocation, which is due to stream discharge below the threshold value in one of its downstream cells. The map shows that cells flowing into smaller streams (right side of the map) are more likely excluded than cells that flow into larger streams (left side of the map). This is due to spatial differences in the relative impact of eucalyptus on stream discharge. Allocating new eucalyptus to a cell will reduce stream discharge in all stream cells downstream of the cell. In the downstream part of the catchment, this has a small relative effect on discharge in the main streams (shown in blue in Figure 3.2A), because streams here are large. As a result, in many realisations stream discharge will not fall below the threshold value in the downstream reaches of the streams, resulting in a low probability for excluding cells from allocation of bioenergy in the downstream region of the catchment. Figure 3.2C gives the map with probabilities of bioenergy in the catchment for the year 2100, which are for the same reason highest in the downstream area. The probability maps are calculated for each year with the `mc_probability` method (see Table 3.3). Figure 3.2D shows the probability of eucalyptus allocation in the selected cell as a time series plot generated by the interactive visualisation of Aguila.

3.5 Discussion and conclusion

In this chapter, we presented the general conditions for a request–reply execution and analysis of coupled component models. We proposed the syntax of function objects as a uniform notation for the modeller to express data requests from external input

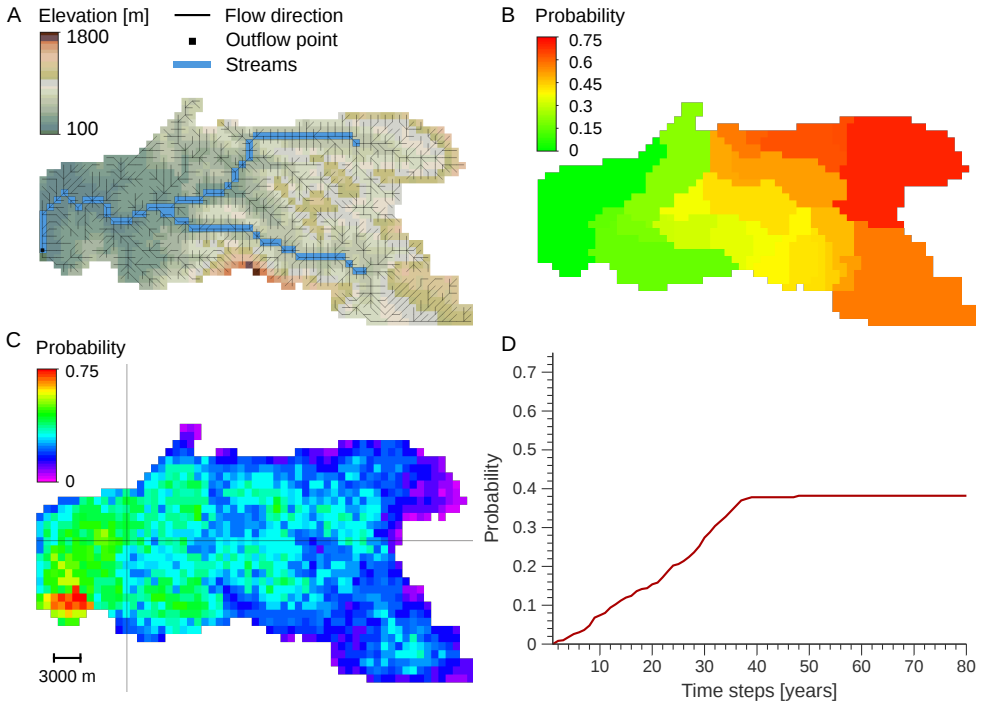

Figure 3.2 (A) Topography, river network and flow direction of the catchment. (B) Probability of the cells being excluded from tree allocation (year 2045). (C) Probability of bioenergy crops in 2100. (D) The corresponding time series plot for the selected cell in panel C.

sources, be it a data file or output from another component model. The developed modelling framework prototype provides a set of classes supporting the implementation of process descriptions supporting a function–based execution of component models. We demonstrated the usage of the modelling framework by means of an integrated model for bioenergy crop allocation and presented results from an uncertainty analysis.

Different operations are in general required to express data import within the transition function, such as an operation to read data from hard disk, or to specify a link to a coupled component model. By using the function notation to call objects providing the requested input, the manner of obtaining the data is transferred from the process implementation to the input interface of a model component. A common notation to express data requests enables a modeller to easier switch between different input sources without modifying the component's process. The process implementation depends less on the way of input data source, therefore making the working of component model more independent and consequently the component more reusable in different modelling contexts. A concise notation for data requirements also resembles the imperative way to specify the transition function f (Equation 3.1): couplings to other components can be expressed in the same way as using intrinsic operations of the modelling environment. Using data request functions allows the modeller to specify explicitly the execution flow of a coupled model. The chronological execution of component models in an integrated model may

therefore be more transparent than in an execution flow automatically generated by a scheduling system (see Chapter 2).

The modelling framework provides with the `reset` section functionality to re–initialise state variables during model runtime. This enables modellers to convert the role of a component model to an accumulator. By specifying aggregating processes, modellers can implement building blocks bridging temporal discretisation differences. We demonstrated the temporal bridging in the case study with the `TotalRunoff` class. Spatial discretisation differences between component models, however, are the second important reason for adapting variables prior to exchange. The modelling framework supports this application case as well allowing to change spatial attributes of component models while executing their process implementations. By using the `setclone` operation in the `dynamic` section, incoming variables can be resampled for each time step to match the spatial discretisation of the outgoing variables.

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

One concern is the efficiency of the current implementation of the modelling framework. Currently, the complete history of state variables is stored to the hard disk, allowing to fulfil arbitrary data requests from any component in the integrated model. Storing the state variables of all component models for each time step and realisation, however, may require significant amounts of disk space as large spatial data sets are involved. In addition, maintaining the history of the state variables introduces a limited organisational overhead. Furthermore, the implementation calculating the yearly runoff (see Table 3.4) also stores the aggregated variable for each time step. As these intermediate values are of no particular use for other component models, storing only the yearly aggregated variable could reduce storage consumption. Potential of further storage reduction could be explored by, for example, in–memory storage of the state variable history. Algorithmic improvements can be implemented discarding the total history in case of equal time steps when only the previous state variable is required.

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

The usage of one single component model class and the autonomy of the component model classes from a specific way of execution allow a modeller to run single component and coupled models in dynamic or Monte Carlo simulations. A modeller can straightforwardly select and assess alternative model compositions, and is therefore assisted in exploratory, plug–and–play modelling using reusable component models (e.g. Papajorgji,

2005; De Kok et al., 2010). The modular architecture of the modelling environment allows adding additional optimisation techniques such as data assimilation (e.g. Evensen, 2003; Moradkhani et al., 2005; Karssenberg et al., 2010). Technically, the `reset` method could also be used by the modelling framework to perform the required state update, when data assimilation approaches such as the Particle Filter are used (e.g. Doucet et al., 2001; van Leeuwen, 2003). Still, observational data may be available for various component models and different time steps. Defining appropriate filter moments in multi–scale component models to understand and to quantify the influence of observational data remains a challenge for an environmental modeller.

4 Bridging discretisation discrepancies

This chapter is based on:
SCHMITZ, O., SALVADORE, E., POELMANS, L., VAN DER KWAST, J. AND KARSSENBERG, D.
A framework to resolve spatio–temporal misalignment in component–based modelling.
In press, Journal of Hydroinformatics.

Abstract

Process–based spatio–temporal component models simulate real world processes, using encapsulated process representations that operate at individual spatial and temporal discretisations. These component models act as building blocks in the construction of multi–disciplinary, multi–scale integrated models. Coupling these independent component models, however, involves aggregation or disaggregation of the exchanged variables at model runtime, since each of the component models exposes potentially different spatial and temporal discretisations. Although conceptual methodologies for spatial and temporal scaling are available, dedicated tools that assist modellers to implement dynamic spatial and temporal scaling operations are rare. We present the accumulator, a programmable general–purpose model building block executing custom scaling operations at model runtime. We therefore characterise runtime information of input and output variables required for the implementation of scaling operations between component models with different discretisations. The accumulator is a component of an integrated modelling framework and can be completed by the modeller with custom operations for spatial and temporal scaling. To illustrate the applicability of the accumulators an integrated model is developed that couples an existing land use change model and hydrological component models at different spatial and temporal scales. The accumulators as building blocks allow modellers to construct multi–scale integrated models in a flexible manner.

4.1 Introduction

Integrated models simulating spatio–temporal changes are important tools for exploring characteristics and interactions of human–natural systems. These models represent real world processes by spatial state transition functions, and provide insight into processes and interactions between system components, which may lead to an improved scientific understanding (e.g. Rotmans, 1990; Liu et al., 2002). In hydrology, integrated models are widely used for river basin management and risk assessment (e.g. Abbott et al., 1986a,b; Jakeman and Letcher, 2003; Ahrends et al., 2008; Argent et al., 2009; de Kok et al., 2009; Karaouzas et al., 2009; de Roo et al., 2011). These models combine knowledge from various domains such as hydrology, ecology, and agricultural sciences (e.g. van Ittersum

et al., 2008; de Kok et al., 2009; Roberts et al., 2010; Janssen et al., 2011; Harvey et al., 2012). The construction of integrated models requires a scientific, semantic and technical integration of individual models (Argent, 2004; Hinkel, 2009; Janssen et al., 2011; Knapen et al., 2013; Elag and Goodall, 2013). Modular construction approaches are valuable in the development cycle of integrated models, as shown by several authors (e.g. Liu et al., 2002; Voinov et al., 2004; Argent, 2005; McIntosh et al., 2005; Papajorgji, 2005; Rizzoli et al., 2008; Castronova and Goodall, 2010; Holzworth et al., 2010; Peckham et al., 2013; Granell et al., 2013a). These studies recommend that environmental modellers describe particular environmental, social, or economic processes in confined component models, and that these component models are coupled to interdisciplinary, integrated models.

Programming languages and several software packages exist that support a modeller in the construction of component models and their integration (see, e.g. Steiniger and Hay, 2009; Jagers, 2010; Granell et al., 2013b). Traditional system programming languages such as Fortran, C or C++ allow for the implementation of virtually any integrated model application from scratch or by combining software libraries. Alternatively, component models can be built by using domain specific modelling languages such as SimuMap (Pullar, 2004), PCRaster (Wesseling et al., 1996), Ocelet (Degenne et al., 2009), or SITE (Schweitzer et al., 2011). These languages are specifically designed for the development of spatio–temporal models in hydrology or other domains. They provide high–level operations on spatio–temporal data types that implement algorithms for common environmental processes, for example, the kinematic wave equation (Chow et al., 1988). Moreover, they provide domain specialists with a comprehensible syntax for model building. Some domain specific languages such as Simile (Muetzelfeldt and Massheder, 2003), STELLA (2013), or ExtendSim (2013) come with graphical modelling languages. Software tools with an emphasis on model coupling such as the Open Modelling Interface (OpenMI, e.g. Moore and Tindall, 2005; Gregersen et al., 2007), the Typed Data Transfer library (TDT, Hinkel, 2009), the Bespoke Framework Generator (Armstrong et al., 2009), or the Model Coupling Toolkit (Warner et al., 2008) provide communication protocols that standardise component interactions. Environmental modelling frameworks such as the Earth System Modelling Framework (e.g. Hill et al., 2004), the Community Surface Dynamics Modeling System (e.g. Peckham et al., 2013), E2 (Argent et al., 2009) or OMS3 (David et al., 2013) provide data types and operations for the construction of model constructs and functionality for their coupling. Finally, software packages that support additional tasks such as data retrieval, data analysis, or visualisation (e.g. Kiehle et al., 2007; Hunter, 2007; Huang et al., 2011; Werner et al., 2013; R Development Core Team, 2013) might need to be integrated in the model development as well.

These software packages can assist modellers both in the technical development phase of individual component models and in their integration. In addition to the problem of choosing the appropriate software tools and the technical challenge of model implementation itself, modellers face a conceptual challenge at the coupling phase as well: given that each individual model has its own specific spatial and temporal discretisations, also called resolution or support size, coupling will result in complex multi–scale integrated models. Modellers are therefore required to adjust the variables that are exchanged between component models such that these variables conform to both

the component model providing the variable and the one receiving the variable. These adjustments can be related to either the spatial discretisation, the temporal discretisation, or both. Adjusting the spatial discretisation is required, for example, when coupling a continental scale coarse grid climate model to a hydrological catchment model that uses smaller grid cells. Similarly, adjusting the temporal discretisation is required when coupling component models that use different time steps. This adjustment of temporal discretisations between component models is a common task in hydrology, where for example highly dynamic processes such as surface runoff are coupled to slower processes such as groundwater flow. An appropriate coupling therefore needs spatial and temporal aggregation or disaggregation of the exchanged variables.

Conceptual methodologies to bridge discrepancies between component models are available as in, for example, common spatial scaling techniques, or methodologies describing temporal aggregation or disaggregation (e.g. Blöschl and Sivapalan, 1995; Bierkens et al., 2000; Skøien et al., 2003; Rastetter et al., 2003; Vermaat et al., 2005; López et al., 2005; Karssenberg, 2006; Barnes et al., 2007; Malone et al., 2012). However, using multiple software packages for the construction of individual component models, and their coupling at various spatial and temporal scales remains a tedious task for a model builder. A modeller, however, can use preliminary alternatives to implement scaling operations. Implementing scaling concepts for dynamic models often remains a manual exercise when using system programming languages. The TDT (Hinkel, 2009), for example, provides support for the transfer of data structures between different programming languages but it does not provide building blocks for upscaling or downscaling these data structures. OpenMI (Moore and Tindall, 2005) provides the model developer with a construct to buffer component outputs and to use these further on with different interpolation techniques (e.g. Elag et al., 2011). The OpenMI framework, however, was designed to couple existing models, and support for the construction of new components models from scratch is thus limited. Other frameworks tailored to the development of integrated models, such as the CSDMS (e.g. Peckham et al., 2013) or ESMF (e.g. Hill et al., 2004) provide coupler components for spatial and temporal interpolation or extrapolation. The implementation of these scaling operations is done on the level of system programming languages. Visual modelling languages such as STELLA (2013), or ExtendSim (2013) provide built–in building blocks for upscaling or downscaling. However, these scaling operations can only be used for lumped models since these software packages have limited intrinsic support for spatial modelling. Spatial explicit operations need to be added using other packages, as was shown by van Deursen and Maes (2008) and Voinov et al. (2004). Stasch et al. (2012) focused on the development of aggregation services for observed data but do not explicitly support runtime aspects of dynamic models. Mennis (2010) proposed a multidimensional map algebra but provided a limited set of built–in operations for spatio–temporal aggregation. Integrated modelling frameworks providing support for both the construction of individual component models and for the implementation of scaling techniques in a suitable way for domain specialists, such as hydrologists or ecologists, are rare.

This chapter describes such a programmable building block for the description of scaling operations, performing aggregation or disaggregation on spatial and temporal

discretisations required in component–based model setups. This is done by including an accumulator building block in a process–based spatio–temporal modelling framework prototype (Chapter 2). This accumulator enables a model builder to describe algorithms on two–dimensional raster data with conventional map algebra operations (Tomlin, 1990), and to describe generic or custom aggregation or disaggregation operations on the temporal discretisation. First, the concepts of the component–based software development practice and its application in hydrological and environmental modelling are briefly introduced. At the same time, the issue of spatio–temporal misalignment between model building blocks occurring at model runtime is raised. Next, the accumulator concept to adjust couplings is shown, and its technical implementation within the modelling framework is outlined. Finally, the application of the accumulator in bridging spatio–temporal discrepancies in hydrological models is illustrated. We report on a first phase of a flood–risk assessment case study for a catchment in Flanders, Belgium. An existing, external land use change model is coupled to a reimplemented spatially distributed hydrological model. The purpose of this case study is to demonstrate the technical implementation of the integrated model, and not to provide model analysis. We conclude with a discussion on noteworthy items in the coupling of multi–scale component models.

4.2 Bridging spatio–temporal discrepancies in modular model setups

The software engineering and integrated modelling disciplines face comparable tasks in the construction of their applications. Both domains involve the construction of complex systems, and they approach this task by assembling modular, reusable, and reliable components into larger systems. This construction process is guided by software engineering practices that organise the structured development of complex software products. Examples are the spiral development method (e.g. Boehm, 1988), extreme programming (e.g. Beck and Andres, 2004), or the more recent agile development strategies (e.g. Cohen et al., 2004; Kniberg, 2011; Rubin, 2012). The objective of these practices is to improve the software development process and product quality. This is achieved by defining tasks and sequencing of software specification, design, implementation, and application. The need to adopt structured development practices in the construction of integrated models is increasingly recognised (e.g. Scheller et al., 2010; Verweij et al., 2010; Schmolke et al., 2010). We now briefly introduce the component–based software engineering practise for the construction of modular hydrological and environmental components. Afterwards, we show its limitations in the coupling of spatio–temporal component models and provide an approach to resolve these limitations.

4.2.1 Component–based construction of integrated models

Component–based software engineering promotes the construction of larger systems out of reusable, independently developed components of confined functionality (e.g. Szyperski, 2002). Independent components can be obtained by the concept of encapsulation where associated functionality is grouped within each component. Components

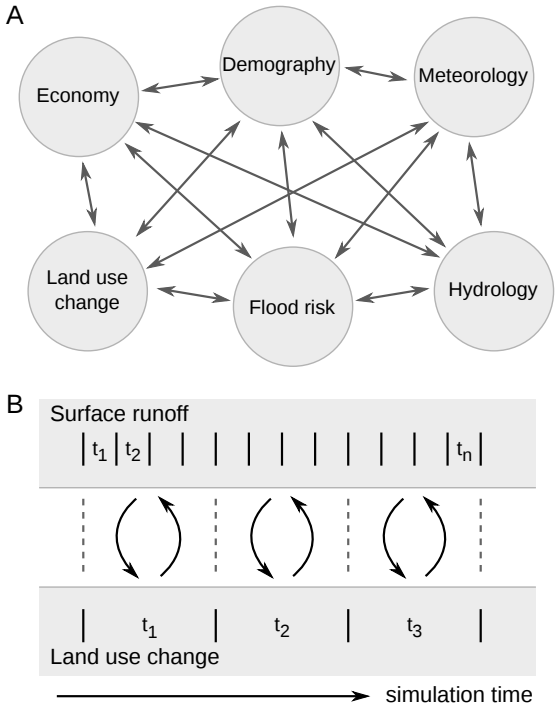

Figure 4.1 Structural and dynamic characteristics of integrated models following component–based development approaches. A) Seamless construction of integrated models enabled by standardised interfaces and a standardised communication protocol. B) Online coupling of component models with different time steps require additional measures to adjust exchanged variables. t_i denote time steps, dashed lines denote exchange periods.

also need to be developed in a decoupled fashion. This implies that no references to other components should be specified in a component to avoid external dependencies. Component–based practises reduce the complexity of larger systems, and enhance maintenance and reliability of modules. Adopting these practices in environmental modelling can lead to the development of generic, reusable components that are deployable in various model applications (e.g. Mineter et al., 2003; Argent et al., 2006; Hinkel, 2009; Donchyts and Jagers, 2010; Voinov and Cerco, 2010). By defining standardised interfaces for the components, and by standardising the exchange of variables, developers can use these components to assemble larger systems in a flexible way. Figure 4.1A shows a conceptual component–based setup of an integrated model. The process representations are encapsulated and different for each component model. The interactions between the component models and the exchange of variables follow a standardised protocol.

Component–based development practices provide approaches for the technical integration of environmental models. Modellers following these practices can map conceptual subsystems into corresponding software components, and they can couple these component models to integrated systems. However, individual component models holding dynamic environmental processes are often associated with characteristic spatial and temporal discretisations (see, e.g. Blöschl and Sivapalan, 1995; Skøien et al., 2003).

When coupling component models with different temporal or spatial characteristics, modellers are confronted to resolve the misalignment (Figure 4.1B). A direct exchange of the variables is not feasible. Modellers therefore need to specify methods for temporal upscaling or downscaling of model variables or parameters to couple the component models.

A potential approach to resolve spatial and temporal discrepancies between component models is to modify the internal process representation. For example, the time step of one model could be adapted such that it matches the temporal discretisation of the coupled counterpart (e.g. Monninkhoff and Kernbach, 2006). This approach is not favourable because of several reasons. First of all, a modification of encapsulated process descriptions places external dependencies on the working of a component model. Adding particular dependencies is against the principle of individual executable components, and results in a less generic applicability of models. Secondly, a process description can be restricted by spatio–temporal constraints of the used numerical solution scheme, or by physically meaningful characteristic spatio–temporal scales. A modification of such processes is therefore undesired. Finally, a modification of process descriptions may not be possible if these are compiled into executable applications. Consequently, aggregation or disaggregation of model variables appears to be the best approach.

Software engineering suggests the adapter or wrapper pattern (e.g. Gamma et al., 1995) to bridge misaligned components without changing the internals of the modules. Thereby, additional code is added that transforms characteristics of output interfaces into input interfaces. The adapter pattern is a commonly used approach in the development of environmental models and in modelling frameworks (e.g. Joppich and Kürschner, 2006; Gregersen et al., 2007; Villa, 2007; Castronova and Goodall, 2010; Becker and Schüttrumpf, 2011; Bulatewicz et al., 2013; OpenMI 2 SDK, 2013; Chapter 5). We extend the adapter pattern to an accumulator building block suitable to bridge component models regarding the spatial and temporal discretisation.

4.2.2 Adjusting spatio–temporal discrepancies

The accumulator is used to spatially or temporally aggregate (upscaling) or disaggregate (downscaling) exchanged variables at model runtime. The accumulator accesses the interfaces of the coupled component models to obtain outputs and to provide adjusted variables, and performs a specific scaling operation implemented by the modeller. To be applicable as a generic model building block, an accumulator needs to implement functionality for different coupling situations. First, the accumulator needs to be able to adjust the spatial discretisation of the exchanged variables. Second, the accumulator should support the exchange of variables between coupled models in both directions. For a coupling of components with smaller time steps to components with larger time steps, a temporal aggregation of variables is required. The contrary direction requires a disaggregation of variables.

Figure 4.2 shows the situation of temporal misalignment between two component models, and introduces the notation that is used in the remainder of the chapter. A component model C_i holds descriptions representing dynamic environmental processes.

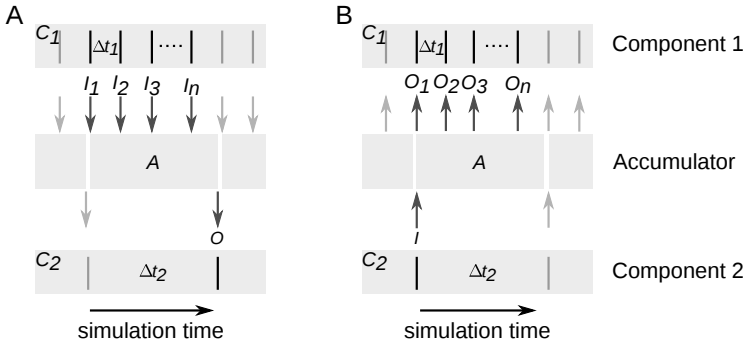

Figure 4.2 Input and output variables considered at an exchange period by an accumulator A. (A) temporal aggregation and (B) temporal disaggregation. Each execution of A is associated with an exchange period consisting of all time indices for input variables I_t and output variables O_t restrained by Δt_2.

These dynamic processes are simulated by iterating a state transition function over a set of discrete time steps Δt_i (Beck et al., 1993; Burrough, 1998; Karssenberg and de Jong, 2005a). Interactions between two components within a time step, as for example in solving differential equations, cannot be represented. The component models are therefore coupled externally (Morita and Yen, 2002).

The exchanged variables represent spatially distributed attributes such as land use classes or hydrological characteristics. The framework uses two–dimensional raster maps with a regular grid as spatial discretisation. The component models provide these maps as output variables for each of these time steps.

Figure 4.2A shows the exchange from component model C_1 with a smaller time step Δt_1 to component model C_2 with larger time step Δt_2. This direction of the exchange requires an aggregation of the variables from C_1 before they are in an acceptable input format for C_2. The accumulator needs to obtain a set of $I_1, ..., I_n$ maps of a variable from C_1, and should provide one output map O to C_2. The modelling framework determines the interval that needs to be considered for the input maps by means of the time step Δt_2 of the component model C_2. The number n of obtained input maps is determined by the time step Δt_1 of C_1.

Figure 4.2B shows the opposite direction of exchange. In this case, a disaggregation of a variable from component model C_2 with time step Δt_2 to component model C_1 with smaller time step Δt_1 is required. The accumulator needs to obtain one input map I from C_2, and to provide a set $O_1, ..., O_n$ output maps in accordance to the time step Δt_1 of C_1.

4.2.3 The accumulator for aggregation

We now formalise the accumulator function such that aggregation and disaggregation can be specified in order to couple component models that run at a different spatial or temporal discretisation. As we focus on the development of integrated models for dynamic environmental systems, we will illustrate the construction of the accumulators

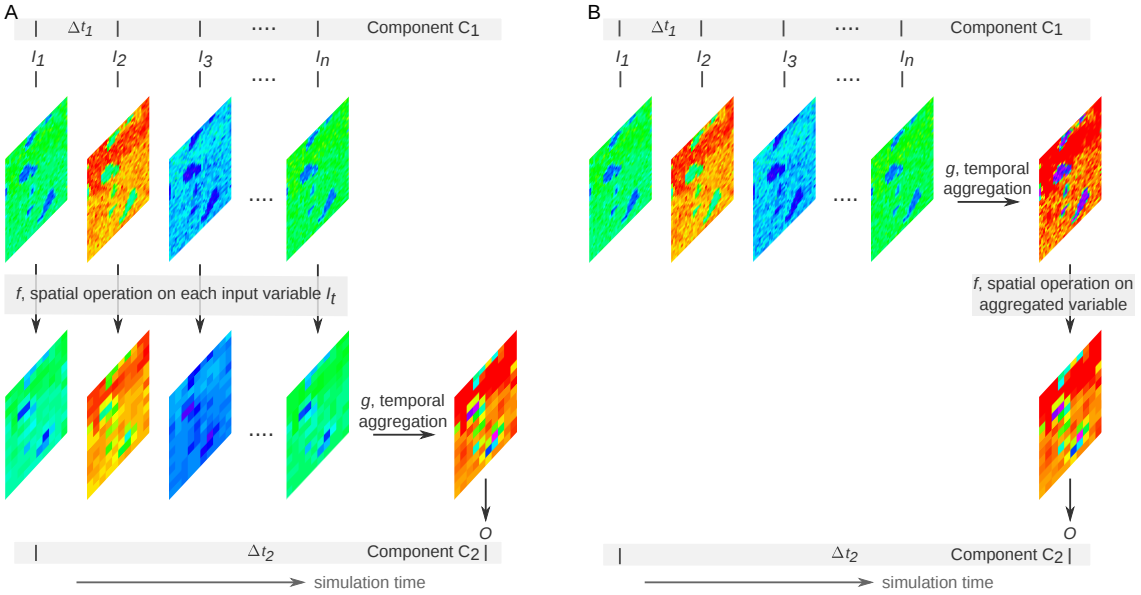

Figure 4.3 Accumulator A (see Figure 4.2A) aggregating two–dimensional variables obtained from a component model C_1 with time step Δt_1. (A) The input maps $I_1, \ldots, I_n$ are first treated spatially (f) and then aggregated temporally (g) to obtain the output O, which is further used by C_2. (B) First executes g, then f. Note that the aggregation (i.e. calculation of O) is done for each time step Δt_2 of C_2.

based on maps with two–dimensional raster data. Comparable concepts will apply to other variable types such as one– or three–dimensional data types.

We first consider an accumulator coupling a component model C_1 with a smaller time step Δt_1 to a component model C_2 with a larger time step (see Figure 4.3). For example, a surface runoff component with a daily time step needs to be coupled to a groundwater flow process component with a monthly time step. Spatially distributed recharge values are output variables of the component model C_1 and are required as input by C_2. To guarantee a flawless data exchange between the building blocks, the modeller needs to be able to adjust the spatial discretisations of maps as well as the temporal discretisation. Adjusting the temporal discretisation requires an aggregation of the recharge values for each month.

To obtain the functional description of an accumulator, we first define f (see Figure 4.3) describing the scaling operation on the spatial discretisation:

$$f : f(I_1, \ldots, I_n) \mapsto O_1, \ldots, O_n \tag{4.1}$$

In Equation 4.1, f describes the spatial aggregation or disaggregation of the exchanged maps, for example, by calculating total or average values for each subcatchment, or by rescaling the spatial discretisation of a raster map. The transformation is performed on each of the individual input map I_t and results in n temporary output maps. For a time step of the groundwater component, for example, in the month January, n corresponds to 31 daily recharge values.

Furthermore, we define a temporal operation g as:

$$g : g(I_1, \ldots, I_n) \mapsto O \tag{4.2}$$

The operation g takes a set of n input maps I_t and aggregates these into one output map O. For temporal operations on raster maps, g is executed for each cell location. Examples for aggregating operations g are logical operations indicating whether a location was affected by phenomena such as flooding or fire, or statistical operations calculating mean discharge values.

With f describing the spatial scaling operations and g describing the temporal scaling operations, the aggregating operation of an accumulator can be described. The accumulator obtains a set of spatio–temporal input maps $I_1, \ldots I_n$ from component model C_i, and provides one aggregated map O to C_j according to the execution of the operation A:

$$A = \begin{cases} g \circ f : g(f(I_1, \ldots, I_n)) \mapsto O \\ f \circ g : f(g(I_1, \ldots, I_n)) \mapsto O \end{cases} \tag{4.3}$$

The aggregated output map O is obtained as a result of a composition of the spatial operation f and the temporal operation g. The modeller can specify the aggregation of input map by two different paths. In the first case described in Equation 4.3, the operations on the spatial discretisation are performed first and then the maps are aggregated over time (Figure 4.3A). In the second case, the aggregation over time is performed first, and afterwards the spatial operation is executed (Figure 4.3B).

4.2.4 The accumulator for disaggregation

The second purpose of the accumulator is to disaggregate variables from larger time steps to smaller time steps. An example of such an application in hydrology is the disaggregation of precipitation data that is only available in a lower resolution than required for high–resolution processes such as surface runoff. A disaggregation is therefore required to cover short term intensity effects (e.g. Güntner et al., 2001). The corresponding accumulator is shown in Figure 4.4, where maps are transferred from component model C_2 with larger time step Δt_2 to a component model C_1 with a smaller time step Δt_1. When disaggregating an output map from a component with a monthly time step to a component with daily time step, for example for the month January, maps for 31 time steps need to be calculated.

Comparable to the aggregation of maps, the modeller can select from two paths to disaggregate a map from C_2 to a set of output maps for C_1. The first option is to expand the temporal discretisation to match the smaller time step by copying the input variable, and then to perform a spatial operation to modify each map of the expanded set (Figure 4.4A). The second option is to first execute the spatial operation, and then to copy this modified map an adequate number of times appropriate for the temporal discretisation of C_1 (Figure 4.4B).

For the operation f modifying the spatial discretisation, we use the same description as given in Equation 4.1. We define an operation h on the temporal dimension expanding

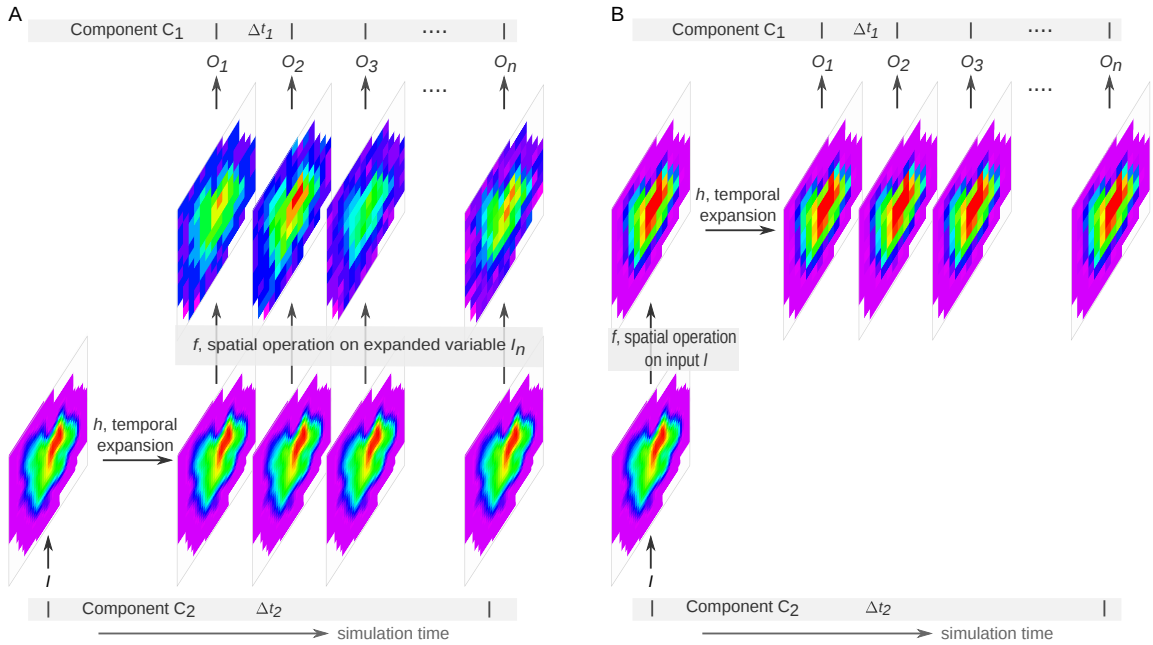

Figure 4.4 Accumulator A disaggregating one input map I from component C_2 to a set of output maps O_i in agreement with the time steps Δt_1 of C_1 (see Figure 4.2B). (A) First executes the temporal operation h, then the spatial operation f. (B) First executes f, then h. The disaggregation is done for each time step Δt_2 of C_2.

one input map I to a set of n output maps O_i according to:

$$h : h(I) \mapsto O_1, \ldots, O_n \tag{4.4}$$

We now can define the function A of an accumulator for disaggregation as:

$$A = \begin{cases} h \circ f : h(f(I)) \mapsto O_1, \ldots, O_n \\ f \circ h : f(h(I)) \mapsto O_1, \ldots, O_n \end{cases} \tag{4.5}$$

4.3 Implementing the accumulators

The concept of a generic accumulator is implemented in a prototype version of an integrated modelling framework. We briefly introduce the framework, and show how a modeller can use the provided accumulator templates to implement operations for spatio–temporal aggregation or disaggregation, respectively.

4.3.1 A modelling framework for component model construction and coupling

The environmental modelling framework introduced in Chapter 2 is used to demonstrate the accumulator concept. The framework provides one environment for the construction of process–based spatio–temporal component models and their coupling. The modelling

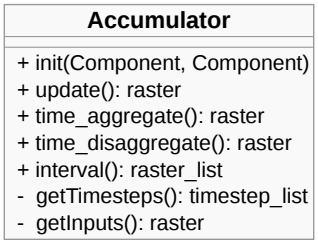

Figure 4.5 UML diagram of the accumulator class provided by the modelling framework.

framework follows component–based software development practices (e.g. Szyperski, 2002; Rizzoli et al., 2008; Argent et al., 2009) and provides component templates with standardised input and output interfaces. The modeller can complete component model templates with descriptions representing natural processes at individual spatial and temporal discretisations. These process representations can be specified by operations building upon the map algebra concept (e.g. Tomlin, 1990; Karssenberg et al., 2007). Accumulator templates can be completed with map algebra operations specifying spatial and temporal scaling. The modelling framework will call the update methods of components and accumulators and hence execute the operations specified by the modeller. The framework includes a management layer that schedules the execution of component models and accumulators. A central instance of the modelling framework maintains a shared time line between all components and derives the execution order based on the current time steps of the model components. The execution of the accumulators is scheduled immediately before a component proceeds to the next time step.

The modelling framework is tailored to domain specialists such as hydrologists or ecologists, who do not necessarily have explicit knowledge in traditional system programming languages. It uses the general–purpose scripting language Python (2013) as model implementation environment. Spatio–temporal operations and data types following the map algebra concept (Karssenberg et al., 2007) and the framework for integrated modelling can be imported as Python modules. If necessary, additional modules for modelling purposes such as libraries for numerical arrays or geospatial data formats can be imported as well (e.g. Langtangen, 2007; Van Der Walt et al., 2011; GDAL Development Team, 2013). By extending the standard Python language with support for spatio–temporal modelling, a modeller can use a flexible modelling environment for the construction of integrated models.

4.3.2 Technical implementation

The accumulator base class

The accumulator is a generic building block that modellers can use to describe spatial or temporal aggregation or disaggregation operations. For this purpose, the framework provides a base class shown in the Unified Modelling Language (Booch et al., 2005) diagram in Figure 4.5. The base class provides public methods that correspond to the functionality introduced in the previous section. The model builder can complete the following two methods. The `time_aggregate` method corresponds to g (Equation 4.2)

and performs the aggregating operations. The `time_disaggregate` method corresponds to h (Equation 4.4) and holds disaggregating operations. The `interval` method enables a modeller to loop over a sequence of spatial variables, and therefore to modify each of the inputs $I_1, \ldots, I_n$ or outputs $O_1, \ldots, O_n$, respectively. The order of the spatial and temporal operations can be specified in the `update` method, corresponding to A (Equation 4.3 and 4.5). The remaining methods shown in Figure 4.5 are used by the modelling framework to obtain the time step information of the component models and the corresponding spatial variables at model runtime.

Obtaining spatial variables at model runtime

At model construction, a modeller specifies in the accumulator how spatial and temporal discretisation differences will be bridged. By calculating the sum or average, for example, a daily variable could be aggregated to a monthly value. The modelling framework executes this accumulator repeatedly at model runtime. It updates the input and output time steps that need to be considered (see Figure 4.2) and therefore organises the correct progress of the accumulator in time.

The accumulator is instantiated with the `init` method. The two arguments are references to the providing component model and the receiving component model, respectively. At model runtime, the accumulator receives the current time step t from the modelling framework and determines the input and output intervals of the coupled component models. In the case of aggregation, the accumulator queries the receiving component model with the `getTimeSteps` method, and obtains the current time step t and the previous time step $t-1$. These time steps are used as interval boundaries to query the providing component model about all available time steps in this interval. For these time steps, the `getInputs` method obtains the spatial variables from the providing component model, and the modeller can write custom code to loop through the spatial variables with the `interval` method. In the case of disaggregation, the time steps t and $t-1$ of the providing component determine the output interval.

Implementing aggregating operations

To illustrate how a modeller can implement the accumulators, we assume a coupling of a rainfall–runoff model with a daily time step to a groundwater model with a monthly time step. Infiltration values need to be transferred from the rainfall–runoff model to the groundwater model. The accumulator provides access to the incoming maps $I_1, \ldots, I_n$, and needs to provide one outgoing map O (see Figure 4.2A). The number of time steps n equals in this scenario the number of days in each month.

To describe aggregating operations, modeller need to implement the `time_aggregate` and `update` methods. Table 4.1 shows the implementation of the aggregating accumulator. In the `time_aggregate` method, a modeller specifies the operation g (Equation 4.2) as custom code. Here, the sum of the infiltration values over one month is calculated for each cell. Line 7 describes the iteration over the daily input maps $I_1, \ldots, I_n$. Lines 8 and 9 describe the operations that are performed on each individual map I_i. First, a spatial operation `scale` (f in Equation 4.1) is rescaling the infiltration map of the current time

Table 4.1 Python script showing an accumulator stub. The custom code is implemented by a modeller and describes an aggregating operation.

```
1 class TotalInfiltration(Accumulator):
2   def __init__(self, inComponent, outComponent):
3     Accumulator.__init__(inComponent, outComponent)
4
5   def time_aggregate(self):
6     sum = scalar(0)                                  ┐
7     for inf in self.interval():                      │
8       dailyInfiltration = self.scale(inf)            │ Custom code
9       sum = sum + dailyInfiltration                  │
10    return sum                                       ┘
11
12  def update(self):
13    return self.time_aggregate()                     ] Custom code
```

Table 4.2 Python script showing an accumulator performing a disaggregating operation.

```
1 class RandomNoise(Accumulator):
2   def __init__(self, inComponent, outComponent):
3     Accumulator.__init__(inComponent, outComponent)
4
5   def time_disaggregate(self):
6     for variable in self.interval():   ┐
7       variable = variable + normal()   ] Custom code
8
9   def update(self):
10    return self.time_disaggregate()    ] Custom code
```

step to match the spatial discretisation of the groundwater model (line 8). Next, the result is added to a map holding the monthly sum. The modeller specifies the accumulator as aggregating accumulator in the `update` method (line 13).

Implementing disaggregating operations

A further application case of the accumulator is to perform disaggregating operations, which are required when coupling component models with larger time step to components with smaller time steps (see Figure 4.2B). The accumulator enables a modeller to access the incoming map I, and needs to provide a set of output maps $O_1, ..., O_n$.

Table 4.2 shows the implementation of the disaggregating accumulator. The modeller needs to implement the `time_disaggregate` (line 5, h in Equation 4.4) and the `update` method. Again, the `interval` method (line 6) enables a modeller to loop over the time steps, in this case modifying the output time steps. The `normal` operation adds random noise to each cell of the incoming map, which is taken here as an example of a simple

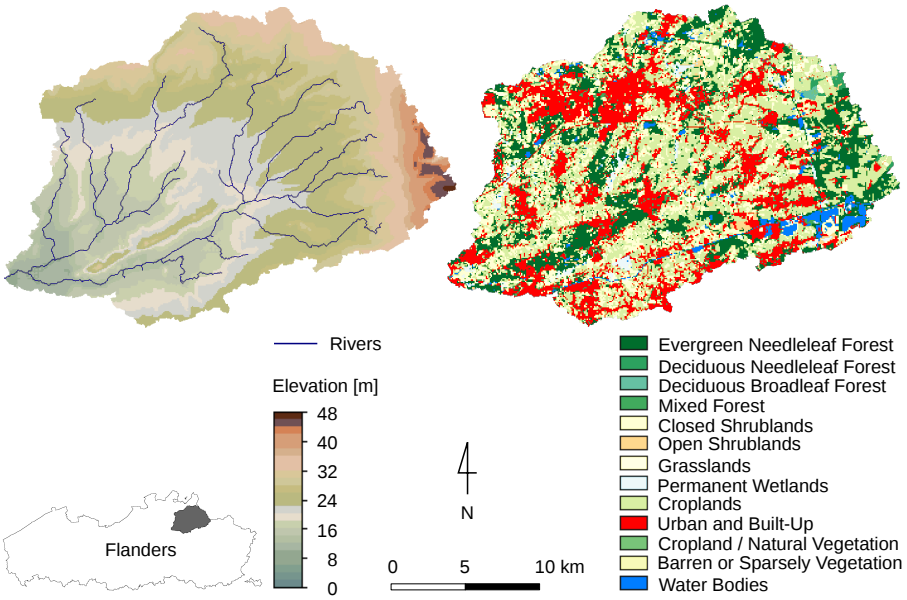

Figure 4.6 Location, topography, river network and land use of the Kleine Nete catchment.

downscaling technique. Additional interpolation techniques that can be implemented by a modeller in the `time_disaggregate` method can be found, for example, in Bierkens et al. (2000) and Elag et al. (2011). The class returns the set of output maps with the call of `time_disaggregate` in the `update` method.

4.4 Application of the modelling framework

To illustrate the usability of the accumulators in the component–based development process of integrated models, we present an instructive proof–of–concept model application for a Belgian catchment. The study focuses on the technical aspects in the coupling of multi–scale component models. The correct execution order of the components and the data exchange at appropriate time steps have been verified (see, e.g. Oreskes et al., 1994; Sargent, 2013). Model evaluation procedures such as calibration and uncertainty analysis (e.g. Refsgaard et al., 2007; Hall and Solomatine, 2008) have not yet been carried out.

A land use change model was coupled to a hydrological model using an existing land use change model for Flanders, based on the MOLAND modelling framework (Barredo et al., 2004; Engelen et al., 2007), and a set of hydrological component models based on the distributed rainfall–runoff model WetSpa (e.g. Wang et al., 1996; Liu et al., 2003).

The integrated model was applied to the Kleine Nete catchment, a 581 km^2 sub–catchment of the Scheldt basin. The catchment is located in the northeastern part of Flanders, Belgium (see Figure 4.6). Elevation ranges between 3 and 48 metres, and the climate is temperate with an average precipitation of 830 mm. The catchment is classified as 60% agricultural area, 20% as forests, and 10% as urbanised area. The average population density is around 380 inhabitants per km^2 (Dams et al., 2012, 2013).

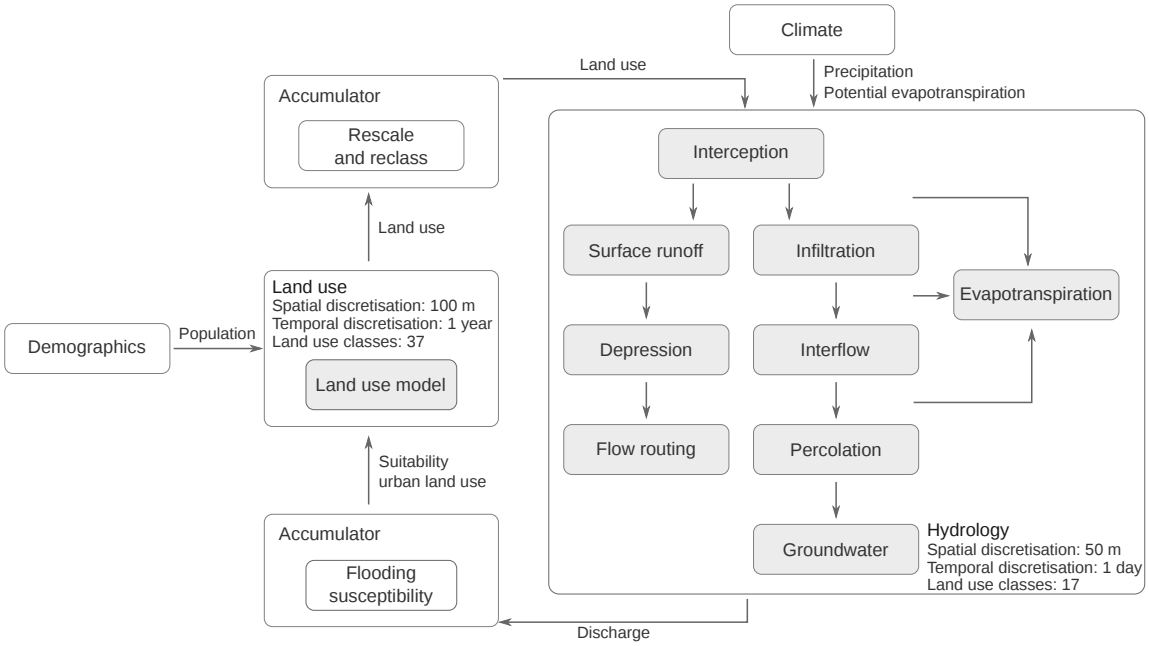

Figure 4.7 Conceptual diagram showing the interactions between the land use change and hydrological component models.

4.4.1 Model structure

The conceptual setup of the model with the main building blocks and interactions is shown in Figure 4.7. Component models are used to model the land use change domain and the hydrological domain. Land use change and hydrology are coupled to data providers supplying population data as time series input, and climate data represented by a set of maps with spatially distributed precipitation and potential evapotranspiration values, respectively.

We choose these two domains for our application, as the evaluation of feedback mechanisms between land use change systems and hydrological systems is relevant and can be used to better analyse and forecast the interactions between land use change and hydrology, in particular to predict and assess impacts of flooding. Feedback mechanisms between land use change and hydrology may lead to an amplification or attenuation of changes. An explicit inclusion of feedback mechanisms is, however, still one of the major challenges in studying land use systems (Claessens et al., 2009). In this application, feedback between the component models is modelled by an exchange of land use maps and flood zone maps. In this way, we evaluate how an increase in sealed surface areas simulated by the land use change model influences surface runoff, and how this affects the risk of flooding. Consecutively, this change in flood risk is used as an input to the land use change model calculating land use change for the next time step taking into account the updated flood risk map.

The land use change component

In this study, a simplified version of the 'RuimteModel' land use model for Flanders is used. RuimteModel is an advanced application of the MOLAND modelling framework (Barredo et al., 2003, 2004; Hagen-Zanker et al., 2005; Engelen et al., 2007) and has been developed to support spatial planners and policy makers in Flanders (Engelen et al., 2005, 2011; de Kok et al., 2012). It is a constrained cellular automata land use model that simulates the likely future development of 37 land use types with a temporal discretisation of one year and a spatial discretisation of one hectare, given alternative planning and policy scenarios and socio–economic trends.

The model input consists of five GIS datasets: the actual land use types, the accessibility to the transport network, the physical suitability for different land use types, the zoning status of the land, and a number of socio–economic characteristics such as population growth and employment in the area. Based on the local interactions with neighbouring land use types, accessibility, physical suitability, and administrative zoning, a transition potential is calculated for each land use type. The land use type with the highest potential is assigned to each of the cells in the study area until the regionally imposed land use demands are met.

For the test case of the Kleine Nete, all socio–economic trends, model parameters and GIS layers, except for the physical suitability, were adopted from the Business–As–Usual (BAU) scenario of the RuimteModel (Engelen et al., 2011). The BAU scenario describes the land use dynamics in Flanders between 2010 and 2050 under the prevailing policies and socio–economic trends. The physical suitability for land use types, on the other hand, is based on the output of the hydrological model and will be updated for each one–year time step of the land use model.

The hydrological components

The WetSpa model, reimplemented in Python by the authors, is used to represent the hydrological processes in the Kleine Nete catchment. The WetSpa model is a spatially distributed rainfall–runoff model for simulations at the catchment scale (e.g. Batelaan et al., 1996; De Smedt et al., 2000; Liu et al., 2003; Safari et al., 2012). The model simulates several physical processes at the level of individual raster cells, such as interception, depression storage, evapotranspiration, runoff, interflow, and groundwater recharge (see Figure 4.7). The main physically–based equations of the model are described in more detail by Liu and De Smedt (2005). Surface runoff is produced using a modified coefficient method based on physical characteristics of each raster cell, i.e. topography, soil type, land use, soil moisture, and rainfall intensity. This method calculates a cell potential rainfall excess coefficient or potential runoff coefficient, having a value between 0 and 1. The coefficient acts as a multiplier of the net precipitation to separate the surface runoff component from the infiltration component. Values of this coefficient are interpolated from literature and adapted to the real physical characteristics of the basin. The calculated runoff is then routed as overland flow and channel flow using the linear diffusive wave approximation method. Interflow is computed based on the Darcy's law and the kinematic wave approximation as a function of the effective hydraulic

conductivity and the hydraulic gradient. The groundwater flow is estimated with the linear reservoir method on small subcatchment scale as a function of groundwater storage and a recession coefficient. A modeller can select the time step discretisation of inputs and outputs from a range of one hour, days, months, or years.

The inputs of the model are: precipitation and potential evapotranspiration time series, which are distributed over the catchment with the Thiessen polygons method if not available in a distributed way such as radar data. Spatially distributed parameters are derived from three base maps using GIS software: topography, soil texture and land use. Outputs of the model are flow hydrographs at the catchment and subcatchment outlets, and maps of evapotranspiration, soil moisture, interception, surface runoff, groundwater recharge, and interflow for each time step or selected periods.

The Python version of the WetSpa model maintains the same capabilities as the original Fortran implementation. However, a different model structure was developed and different GIS software (PCRaster, 2013) was used. The new structure is modular, allows to receive land use as a dynamic input variable, and every physically–based process is coded in a separate component model. These component models are coupled and exchange data at run time through the modelling framework prototype (see Figure 4.7). Existing case studies were used to verify the correctness of the reimplementation.

For the test case of the Kleine Nete the topography, resampled to 50 m from the original 25 m resolution digital elevation model of Flanders (OC GIS-Vlaanderen, 2001), is used as GIS input map. Meteorological data of rainfall and potential evapotranspiration were obtained from the Royal Meteorological Institute of Belgium. The temporal discretisation was set to a one–day time step.

Flood risk assessment

To model the feedback from the hydrological component models, we incorporate the effects of the stream discharge in the physical suitability for urban land use classes. We modified the input variable holding the physical suitability for urban land use classes and decreased the suitability for housing because of an increased flood risk in a certain cell.

Water levels are derived from the discharge values with a stage–discharge relation. The water levels are translated to a two–dimensional water surface, and cells were identified as being flooded when the difference between the water surface and the elevation map exceeds a certain flooding threshold. The flood zones are used to determine the suitability for urban areas. Flooded cells are assigned as not suitable for housing to have a clear view on the effects of the bidirectional coupling.

The calculation of the flood zones follows the approaches described for example by Ward et al. (2011) or Haasnoot et al. (2012). In the first stage of the model construction, we tentatively assigned a binary value to determine the suitability for housing. This implementation could be improved by adopting a more rigorous approach, assigning for example the suitability based on damage cost functions (e.g. ICPR, 2001; de Kok and Grossmann, 2010).

Table 4.3 Python script specifying components and interactions of the integrated model for flood risk assessment. Not all instantiations of component models and interactions are shown.

```
1 # Creating instances of the component models
2 luc = Luc(datetime(2010,1,1), datetime(2030,1,1), timedelta(days=365))
3 hyd = Hyd(datetime(2010,1,1), datetime(2030,1,1), timedelta(days=1))
4 demog = Demog(datetime(2010,1,1), datetime(2030,1,1), timedelta(days=365))
5
6 # Add components to instance of the integrated model
7 caseStudy += luc
8 caseStudy += hyd
9 caseStudy += demog
10
11 # Specify interactions
12 caseStudy.link(demog["population"], luc["population"])
13 adapt = ReclassResample(luc["landuse"], hyd["landuse"])
14 caseStudy += adapt
15 caseStudy.run()
```

4.4.2 Model implementation

Land use change is modelled by wrapping the RuimteModel into a component model of the modelling framework. The hydrological component models are implemented in Python and therefore integrate seamlessly into the modelling framework. More examples of component models can be found in Chapter 2.

Table 4.3 shows an excerpt of the model algebra script specifying the components and interactions of the integrated model for flood risk assessment. First, the individual components are instantiated with their simulation period and time steps (lines 2–4). The script shows two component models `Luc` and `Hyd` modelling land use change and rainfall–runoff, respectively, and a component `Demog` providing population data. All components are then added to an instance of the integrated model (lines 7–9). Next, the modeller specifies the exchange of variables between the component models. A direct transfer of variables without temporal aggregation is shown in line 12 where the population output variable of the demographic data provider is coupled to the corresponding input variable of the land use change component. An accumulator `ReclassResample` needs to interconnect the two component models because the characteristics of the outgoing variables are not consistent with the characteristics of the incoming variables (line 13).

Integration of external functionality into a modelling framework

Several use cases can require the embedding of an external application into a modelling framework. The embedding is advantageous if the domain to be integrated is outside of the profession of the modeller, and required if the source code of an application cannot be modified. In addition, a time consuming reimplementation and testing of new model code can be avoided by integrating a previously applied model application. We use the

Table 4.4 Python implementation of the component that models land use change. The external RuimteModel is executed by a system call. The initialisation of some variables is omitted.

```
class LucModel(PCRasterRealTimeComponent.PCRasterRealTimeComponent):
  def __init__(self, start, end, deltaT, cloneMap):
    PCRasterRealTimeComponent.PCRasterRealTimeComponent.__init__(self, cloneMap, start, end, deltaT)
    self.landuse = self.readmap(cloneMap)
    self.suitability = self.readmap(cloneMap)
    self.addInputValue(self.suitability)
    self.addOutputValue(self.landuse)
    # specify fixed path settings
    self.workingDir = os.getcwd()
    self.exePath = os.path.join(self.workingDir, "bin", "ruimtemodel.exe")

  def runTimestep(self):
    # write new suitability map to input directory of land use change model
    self._writeSuitability(self.suitability)
    # generate command line string
    currentYear = self.timeStepAsRealTime().year
    landuseInputPath = os.path.join(self.workingDir, "kleine_nete", "lu_%d.asc" % (currentYear))
    landuseOutputPath = os.path.join(self.workingDir, "kleine_nete", "lu_%d.asc" % (currentYear + 1))
    cmd = "%s %s %d %s %s %f %d %s" % (self.exePath, self.simFilePath, currentYear, landuseInputPath, ruleFilePath, self.alpha, currentYear + 1, landuseOutputPath)
    self._executeCommand(cmd)
    # transfer land use change model results to process component output variable
    cmd = "gdal_translate -of PCRaster -ot Int32 -mo PCRASTER_VALUESCALE=VS_NOMINAL -a_nodata 37 %s %s" % (self.rstPath, self.pcrPath)
    self._executeCommand(cmd)
    self.landuse = self.readmap(self.pcrPath)
```

wrapper approach to integrate the existing land use model, thereby obtaining a reusable land use change component complying with the modelling framework. Wrapping allows to use the component model in other case studies, for example to investigate the influence of land use change on the economic quantification of ecosystem services (e.g. Kragt et al., 2011; Broekx et al., 2013).

A component model therefore acts as a proxy for the embedded model application (Hahn et al., 2009). However, a model application needs to fulfil certain requirements to be embedded into a modelling framework. First of all, the application must be able to proceed a number of time steps stipulated by the modelling framework. Secondly,

Table 4.5 Python script showing the accumulator with operations converting the number of land use classes and operations rescaling the grid cell size of the land use variable.

```
1 class ReclassResample(Accumulator):
2   def __init__(self, inComponent, outComponent):
3     Accumulator.__init__(inComponent, outComponent)
4
5   def time_aggregate(self):
6     pass
7
8   def update(self):
9     variable = self.interval()
10    landuse = lookupnominal("luConversion.tbl", variable)
11    landuse = rescale(variable, 0.5)
12    return landuse
```

all required inputs and outputs of the external application must be accessible for a modelling framework, for example by file exchange via hard disk or through a command line interface. Finally, the application must not require user interaction such that the framework can execute the application automatically.

The land use model used in this case study complies with these requirements, and its land use change processes are mapped to the component model `LucModel`. The Python script implemented by the modeller shows the wrapping of the RuimteModel (Table 4.4). The `init` section specifies the temporal characteristics such as start and end time as well as the time step of the component (line 3). In addition, the state variables such as the land use map are initialised, and the input and output variables and therefore the component interface are specified. The land use model requires a specific folder structure, such as folders for required libraries, input and output files. The lines 9–10 construct path locations referring to these static elements such as the model executable, or the scenario input files.

The `runTimestep` section is executed for each time step. First, the land use change model is initialised with the updated inputs of the `LucModel` component. Therefore, the incoming spatial suitability map is written to the input folder of the RuimteModel (line 14). Second, the external application needs to be executed for a one–year time step. The lines 16–20 describe the construction of the command line string carrying out the execution of the land use model. The input file location of the current land use for time step t and the output file location for time step $t+1$ are constructed. The command line string is composed of the executable and its input arguments, specifying input and output file locations and the simulation period. Finally, the new land uses calculated by the land use model need to be provided as output variables of the `LucModel` component. The lines 22–24 show this conversion using the GDAL library (2013).

Table 4.6 Python implementation of the accumulator interconnecting the hydrological components and the land use change component. The symbol | represents the spatial Boolean OR.

```
1 class FloodingSusceptibility(Accumulator):
2   def __init__(self, componentIn, componentOut, variable):
3     Accumulator.__init__(componentIn, componentOut, variable)
4
5   def time_aggregate(self, variable):
6     floodTotal = boolean(0)
7     for discharge in interval():
8       waterSurface = self.getWaterSurface(discharge)
9       depth = ifthen(waterSurface > dem, waterSurface - dem)
10      flood = clump(ifthen(defined(depth), 1))
11      floodTotal = floodTotal | flood
12    return self.scale(floodTotal)
13
14  def update(self):
15    return self.time_aggregate()
```

Implementing accumulators with spatio–temporal scaling operations

Since the land use model and the hydrological component models use different spatial and temporal discretisations, a direct exchange of maps between these domains is not feasible. The land use model uses 37 land use classes, a 100 m grid cell size, and yearly time steps. The hydrological components use 17 land use classes, a 50 m grid cell size, and daily time steps. Accumulators are therefore required in both directions for an appropriate coupling of the two domains.

Table 4.5 shows the accumulator `ReclassResample` adjusting the land use output map for each year to an adequate input map for the hydrological model (see Figure 4.7). The accumulator is initalised with the references to the providing and receiving components (see Table 4.3) including the exchanged variable, in this case the land use classes (line 3). The temporal operation g (Equation 4.2) is not implemented in the accumulator (`pass` in line 6). The `update` method only performs operations on the spatial discretisation. The spatial scaling (f in Equation 4.1) of the land use variable is implemented with operations from the PCRaster Python module (Karssenberg et al., 2007). The 17 new land use classes are assigned by the `lookupnominal` operation based on the conversion table given in the file `lucConversion` (line 10). The cell size is modified by `rescale` with a scaling factor of 0.5, directly assigning the land use classes to the 50 m grid used by WetSpa (line 11). The `update` operation returns a land use map matching the discretisations of the hydrological components.

The transfer of the output maps from the hydrological components to the land use change component model requires an aggregation of the daily discharge values to a yearly value in addition to the spatial scaling. Table 4.6 shows the implementation of the corresponding accumulator. The calculation of flood zones for each year is implemented in the `time_aggregate` method (g in Equation 4.2). For each day of the expired year, the

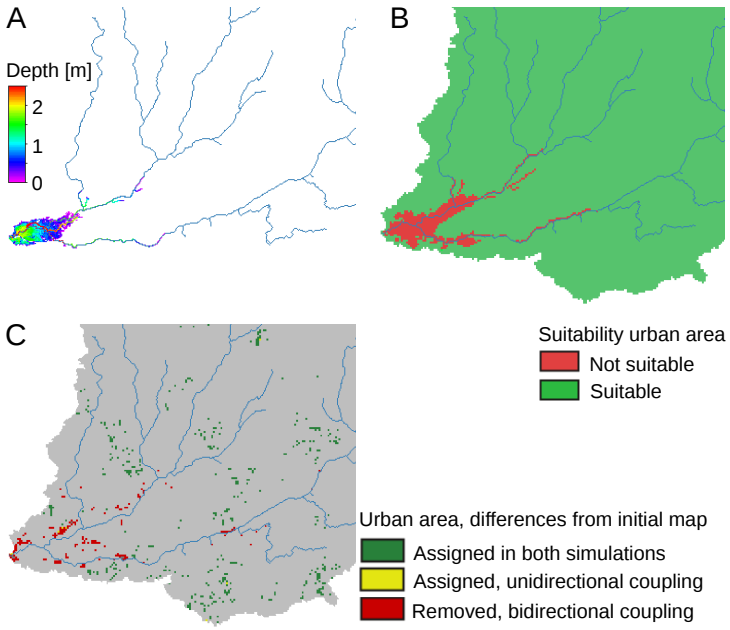

Figure 4.8 Results of the integrated model at different time steps. (A) Inundation depths, derived from a daily discharge value. (B) Urban suitability, derived from accumulated flooding zones over one year. (C) Allocation of urban areas after two ten–year simulations with different interactions between land use change and hydrology.

discharge value is converted to a water level and subsequently to a water surface map (line 8). The difference with the elevation map determines the inundation depth for each cell (line 9). The `clump` operation groups all affected cells into a flood zone. The result is a map holding Boolean `True` values in cells that were inundated in the current time step. The Boolean `OR` operation (line 11) combines the flooded cells of the current time step with the yearly total flood zones. The result is scaled to a 100 m discretisation before it is transferred to the land use change component.

Figure 4.8 shows output maps from the component models and accumulators. Note that these outputs are preliminary results obtained with a technically verified but uncalibrated model. The results are only included to demonstrate possible outputs of the integrated model. Figure 4.8A shows the inundation depths occurring at one day. Figure 4.8B shows a suitability map as a result of the aggregated flood zones of the last year. Cells affected by flooding are marked as not suitable for urban development. Figure 4.8C shows the differences in the newly allocated urban land use classes for a 10–year simulation. When simulating using a unidirectional coupling, in which only the land use in the hydrological component model is updated, only a few cells in the south–west of the study area are transformed to urban area. Incorporating a bidirectional exchange between the land use and the hydrological component models leads to a disappearance of cells classified as urban in the area susceptible to flooding.

4.5 Discussion and conclusion

Component–based development practices applied to environmental modelling can lead to a more straightforward construction of generic and reusable component models. Coupling multi–scale component models and runtime interactions at model execution, however, demand a modeller to adjust exchanged variables for a sound setup of integrated models. The presented accumulator is a generic model building block suitable to hold these adjustments. The accumulator provides a tool to implement aggregation or disaggregation operations on spatial and temporal discretisations, which can be completed by the model builder using map algebra operations. The usage of the accumulator was illustrated in a case study by constructing an integrated model for flood risk assessment consisting of several component models running at different spatial and temporal discretisations.

The component–based development practices and modular development approaches help to limit the complexity of large systems by representing natural processes with confined process representations. The design and development can be carried out without affecting other component models, a facet becoming more important in maintaining a longer lifecycle of component models (e.g. Scholten et al., 2007; Hahn et al., 2009). The need to implement spatial and temporal scaling operations in the construction of multidisciplinary models will increase when component–based model development practices are used. However, approaches that provide generic model building blocks for scaling operations that are programmable by modellers just like component models are rare. Modellers using system programming languages for model development or scripting languages as glue for component model integration (e.g. Roberts et al., 2010; Huang et al., 2012) receive no built–in support for time management, distribution of exchanged variables, and implementation of scaling operations themselves. Frameworks with emphasis on the component coupling such as TDT (Hinkel, 2009) or OpenMI 1.4 (e.g. Gregersen et al., 2007) focus on the specification on the component interfaces and can require a modification of the component interfaces or require a requested component to perform conversions (Moore and Tindall, 2005). Additional dependencies can therefore be imposed on individual component models, making them less reusable in other model applications. Hence, the latest version of the OpenMI standard (e.g. Donchyts et al., 2010) introduces the IAdaptedOutput interface suitable for spatial scaling or unit conversions to overcome this requirement. Building blocks for numerical integration or extrapolation that are provided in graphical modelling systems such as ExtendSim (2013) lack support for spatial explicit operations. Environmental modelling frameworks such as the ESMF (e.g. Hill et al., 2004) or CSDMS (e.g. Peckham et al., 2013) allow modellers to implement scaling operations but rather on the level of programming languages than on domain concepts. The modelling framework presented in this study is an important addition to these existing frameworks, particularly because it provides a single research environment containing building blocks to implement component models, couplings, and scaling operations. Calibration and data assimilation techniques can be added to the same framework comparable to Karssenberg et al. (2010). The result is a single software environment that supports almost all steps in integrated model construction. Modellers

can therefore stay within the same software environment during model construction, which minimises the time required to invest in learning how to use the tools (Killcoyne and Boyle, 2009).

A modeller can use the introduced accumulators to implement spatial and temporal aggregation or disaggregation operations, allowing for a runtime coupling of component models associated with their characteristic scales. The accumulators offer the modeller only a technical means to straightforwardly access variables required for implementing the scaling operations of coupled component models. Model builders, however, still need to regard characteristics of component models and the designated scaling operations while establishing a coupling. Obtaining knowledge about the component models is required as those may have internal space–time constraints that restrict couplings and scaling operations. The influence of parameters—obtained by calibrating the component models—on the integrated model needs to be evaluated as well. The MOLAND–based land use model and WetSpa were used as individual models in several scientific studies (e.g. White et al., 1997; Liu and De Smedt, 2005; Engelen et al., 2007; Hurkens et al., 2008; Poelmans et al., 2010; Safari et al., 2012) but an appropriate assessment of the parameter values on the coupled model still needs to be carried out. In addition, the methodology used in an accumulator strongly influences the outcomes of the integrated models. Scaling operations might introduce errors. Aggregation, for example, inevitably causes the loss of information (e.g. van Beurden and Douven, 1999). In addition, the order of operations on space and time given by the modeller will produce different results in non–linear scaling schemes. Mass balance properties need to be maintained during disaggregation, for instance when disaggregating monthly precipitation values into daily values. Moreover, conceptual problems may arise in the coupling of component models from domains with a strong spatial modelling background such as hydrology to domains without a traditional explicit treatment of space such as economics. Finally, the introduced accumulators do not yet provide a set of standard operations calculating statistics over time on raster–based maps, for example operations such as timeaverage (Karssenberg and de Jong, 2005b). However, this limitation is a result of the prototype implementation status of the modelling framework rather than a conceptual limitation.

The framework implementation requires a component model to store the complete history of state variables to the hard disk. Therefore, arbitrary data requests can be fulfilled from any other component. Storing the state variables of all component models for each time step, however, can require significant amounts of disk space when large spatial data sets are involved. Potential storage reduction could be explored by in–memory storage of state variables, similar to the SmartBuffer in OpenMI (see, e.g. Elag et al., 2011).

So far, only a few studies quantify the effects involved in the coupling of multi–scale integrated models (e.g. Claessens et al., 2009; Moreira et al., 2009; Elag et al., 2011). Further research is needed to assess the influence of the time step of individual component models, and the choice of aggregation and disaggregation over space and time on feedback effects. We believe that the programmable accumulator provided with our modelling framework can help a modeller in the assessment of alternative couplings of integrated models.

5 Integrating external model applications

This chapter is based on:
SCHMITZ, O., KARSSENBERG, D., VAN DEURSEN, W. P. A. AND WESSELING, C. G. (2009), Linking external components to a spatio–temporal modelling framework: Coupling MODFLOW and PCRaster. Environmental Modelling & Software 24 (9), 1088–1099.

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.

5.1 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 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 chapter 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, 2005a; Burrough, 1998). In dynamic modelling, which is the term used here, the state of a model is iteratively updated over a set of discretised time steps 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. Leavesley et al., 1998; Wang and Pullar, 2005; Hurkens et al., 2008; PCRaster, 2013; Repast, 2013; MATLAB, 2013). For model construction, three features of a modelling framework are essential (Karssenberg, 2002; Argent, 2004): 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, 2013) to high level approaches that provide operations representing spatial processes on 2D and 3D entities representing a landscape (Pullar, 2004; PCRaster, 2013; ArcGIS, 2013). 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, 2013; GRASS, 2013) 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, 2013). 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 time steps. 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 (Gregersen et al., 2007; OpenMI, 2013).

In this chapter we describe and evaluate the PCRaster modelling framework (van Deursen, 1995; Karssenberg and de Jong, 2005a; PCRaster, 2013). 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 (2013). 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 chapter 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 useability for model construction whereby we discuss the use of OpenMI as an alternative strategy to link external components.

5.2 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 time step t_i:

$$Z_{1,\ldots,m}(t_i) = f(Z_{1,\ldots,m}(t_{i-1}), I_{1,\ldots,n}(t_i), P_{1,\ldots,l}) \quad \forall\ t_i \tag{5.1}$$

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, 2005a). Arguments to that function are a set of parameters $P_{1,\ldots,l}$ and input parameters $I_{1,\ldots,n}$ of the current time step. 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 time step. 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 5.1).

5.3 PCRaster modelling framework

PCRaster (2013) 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 (Tomlin, 1990; Karssenberg and de Jong, 2005a) (see Figure 2.2):

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 (Figure 5.2).

In contrast to most GISs, PCRaster provides capabilities for modelling in space and time. Figure 5.1 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., 2010), geomorphologic modelling (Karssenberg and Bridge, 2008) or wind erosion modelling

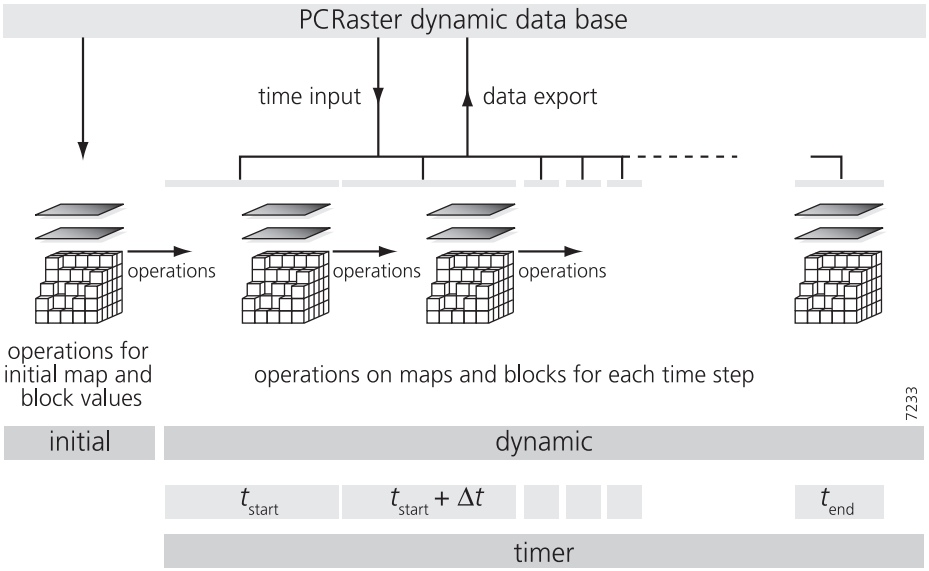

Figure 5.1 Conceptual overview of a PCRaster model run. The initial section is used to establish valid starting conditions, the dynamic section performs the process descriptions for each time step.

(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).

5.3.1 The PCRaster PCRcalc modelling language

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 5.1. 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 5.1) 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:

```
resultMap = nativeOperation(argumentMaps);
```

The combination of several statements allows the formal representation of the processes occurring in a system, as shown in the example of Table 5.1. Here the initial section is used to create a local drain direction map as shown in Figure 5.2 containing the flow

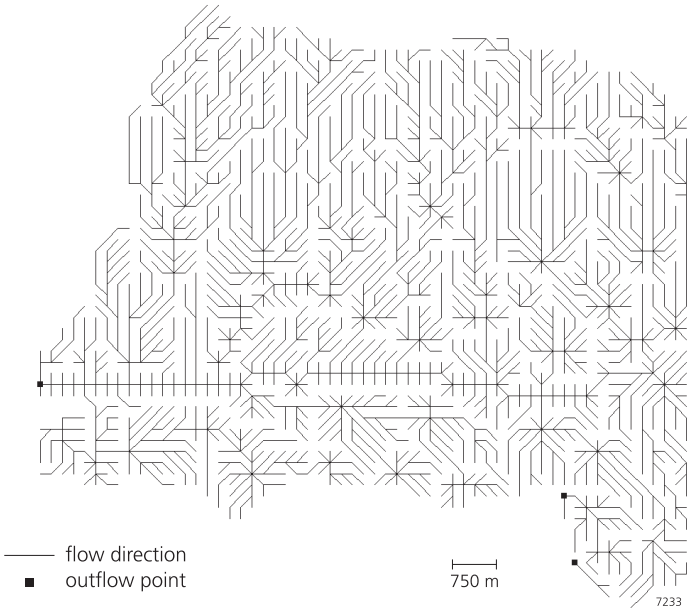

Figure 5.2 Local drain direction network map of a catchment. Flow directions are indicated from each cell to its steepest downslope neighbour. The outlet is located at the left side of the catchment.

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.

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 5.2 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.

Table 5.1 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 5.2 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>
```

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 5.3 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.

Table 5.3 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;
}
```

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 5.4.2.

5.3.2 The PCRaster Python language

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.

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 (2013). 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

Table 5.4 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()
```

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 5.4 shows the Python version of the PCRcalc runoff model given in Table 5.1 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.

5.4 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.

5.4.1 Linking the MODFLOW component

MODFLOW (Harbaugh et al., 2000), developed by the US Geologic Survey (USGS, 2013) 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 (2013).

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; Carrera-Hernández and Gaskin, 2006; Processing Modflow, 2013; VisualMODFLOW, 2013). 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 time steps 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 5.3.1 and 5.3.2. The extension updates necessary MODFLOW input files each time step 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.

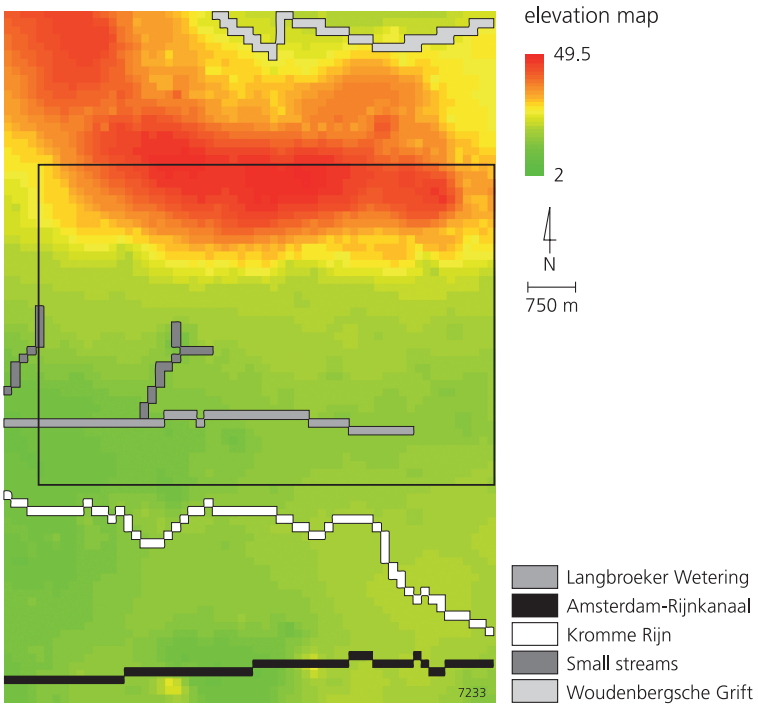

Figure 5.3 Map showing elevation values and streams. The major stream in the south is the Amsterdam–Rhine canal with an average width of 150 m, the smaller streams have an average width of 10 m. The box indicates the location of the basin shown in Figure 5.2.

5.4.2 PCRaster Modflow

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 5.5 and continued in Table 5.6. For the PCRaster Python version of the same model we refer to Table 5.7 in Appendix 5.A.

Figure 5.3 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 5 000 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.

Initial model section

As shown in the model script in Table 5.5 and Table 5.6, 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.

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

Table 5.5 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);
```

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 5.6. 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 Figure 5.4.

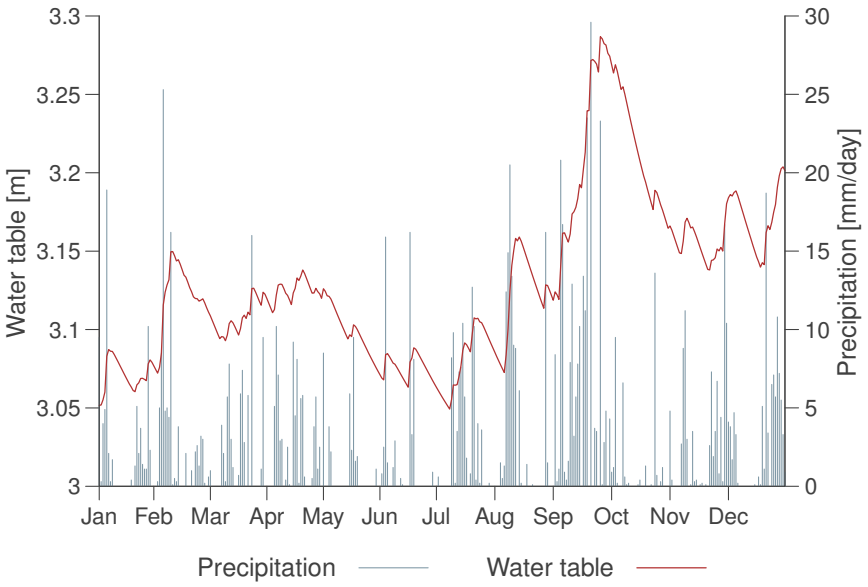

Figure 5.4 Response of the water table to the effective precipitation for one specific cell in the catchment.

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 Figure 5.3 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 Figure 5.5.

5.5 Discussion and conclusion

In this chapter 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.

Table 5.6 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));
```

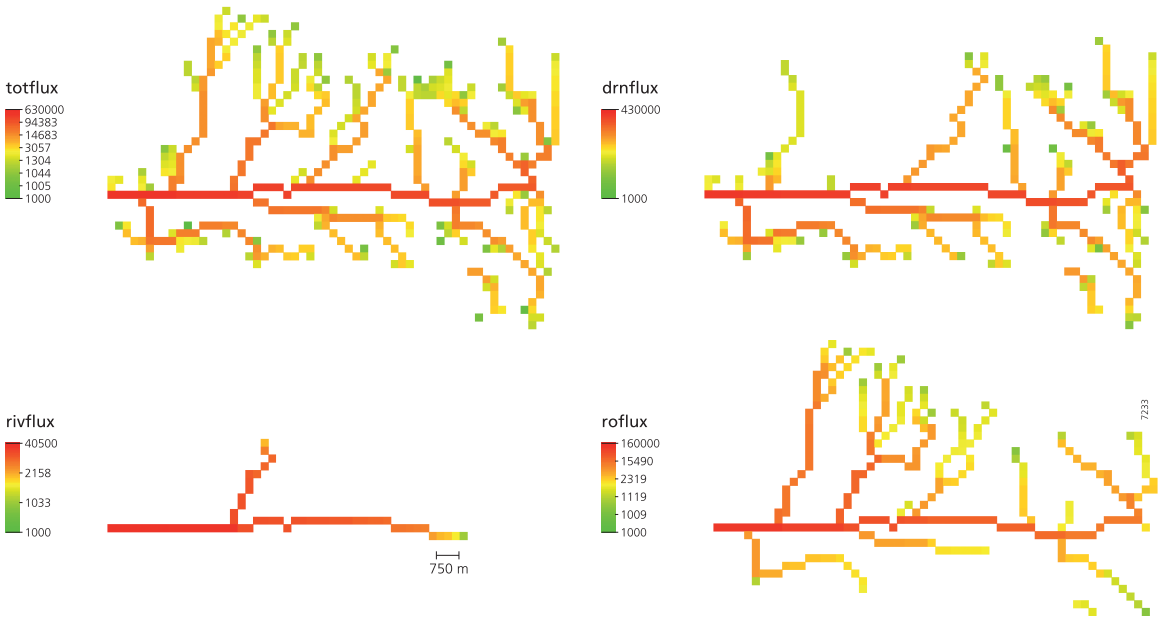

Figure 5.5 Simulated total and separate fluxes in a subcatchment for September, 15 (timestep 258) after a strong rainfall event. Units in m^3/day.

In general, the selection of a link strategy between modelling components depends on the type and quantity of the linked components. Figure 5.6 shows three different groups of external model components. The groups shown in panel (a) and (b) of Figure 5.6 can be linked to the modelling framework using the approach described in this chapter. Figure 5.6(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 time step. 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 5.3.1.

Figure 5.6(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 (Figure 5.6 a 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. Figure 5.6(c) shows a situation of integrated modelling that is difficult to handle with the linking approach described in this chapter. In this situation, a large number of model components need to be integrated, where it is also difficult to map their time steps to the time steps of the modelling framework. Here, it is better to use a modelling interface such as OpenMI (2013). OpenMI is a communication standard for linking modelling frameworks. Each

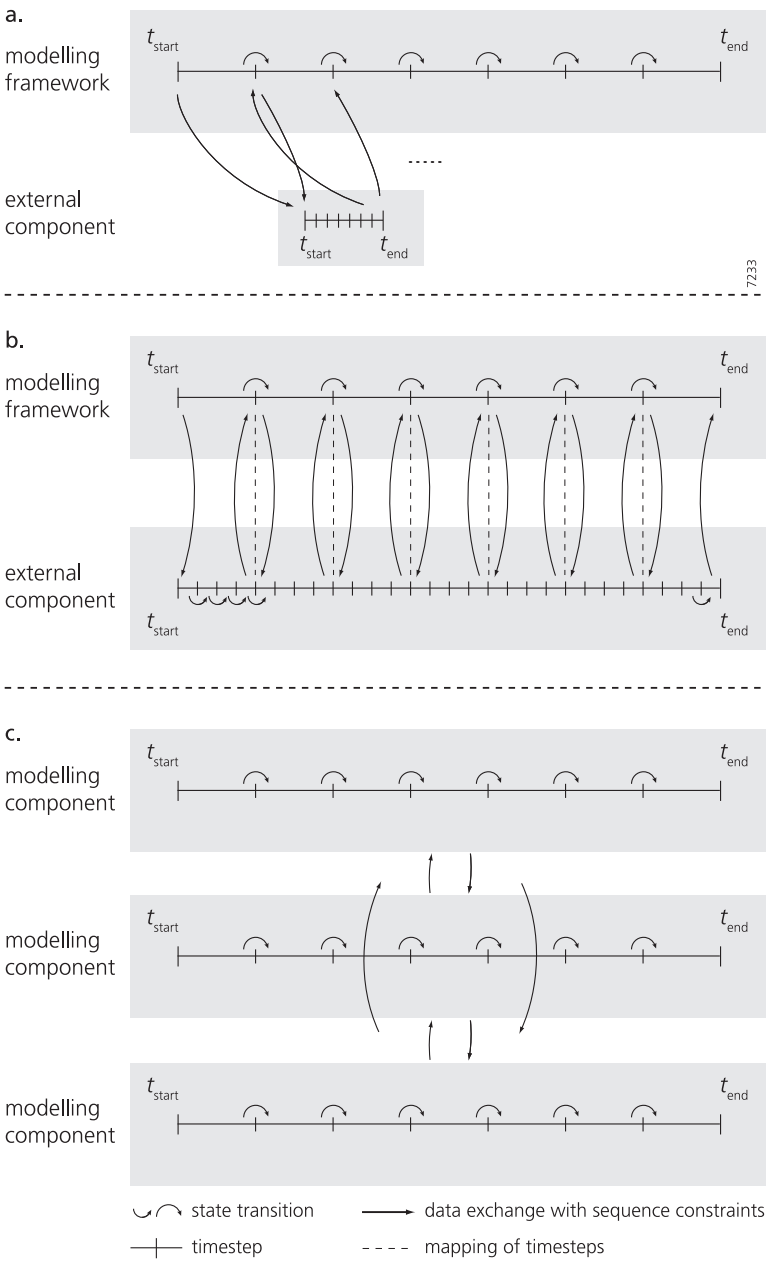

Figure 5.6 Different approaches of linking model components. (a) Linking a single operation (b) Link to an independent modelling component (c) Arbitrary linking of modelling components.

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 reuse of sub–components and a better overview of complex models.

The framework presented in this chapter 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 (2013) allow to model temporal dependencies, but do not support to represent spatial interactions and processes. Matlab (2013) 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 (2013) 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 an 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).

5.A PCRaster Python example script

The following code shows the Python version of the PCRcalc script given in the case study of section 5.4. The *binding, areamap* and *timer* sections are no longer necessary. Names and argument types of the PCRasterModflow operations equal the PCRcalc version.

Table 5.7 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 seapage
    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()
```

6 Semantic annotation of integrated models

This chapter is based on:
SCHMITZ, O., DE KOK, J.-L., DE JONG, K. AND KARSSENBERG, D.
Formalising component interfaces for building integrated models. In preparation.

Abstract

The challenges in developing geoscientific computer models are the construction of modules representing individual processes, and the assembly of these modules into integrated systems. Existing environmental modelling languages support modellers in the construction of generic model building blocks. Coupling of these blocks, however, still occurs on the level of programming languages. Consequently, scientists as model builder lack a manner to distinguish the programming code and the domain specific meaning captured by the models. This hampers standardised coupling, reuse, and archiving of model building blocks.

To overcome this problem, we propose an implementation–independent formalisation describing the semantics of integrated models. Tailored to spatio–temporal process–based models, we define process components, accumulators, and data providers as key building blocks. We identify and formalise invariant characteristics required to describe the spatial and temporal context of these building blocks, and coupling of these building blocks. The semantic content is used to validate components and couplings to prevent the exchange of mismatching information. To qualify as a tool for domain specialists we enhance an existing declarative modelling framework with a semantic annotation layer. Semantic enrichment of model building blocks consolidates their application in integrated systems and is a step towards a simplified exchange of model constructs.

6.1 Introduction

The software engineering discipline emphasises the need for software development practices that provide mechanisms supporting reuse of code. A widely used practice that follows this approach is component–based software development (e.g. Szyperski, 2002; Booch et al., 2007), which uses the concepts of interoperability and integration to support code reuse. The interoperability concept implies that code is written with the intention of reuse, which is achieved by modules of limited complexity adhering to a common interface. Limiting the complexity of software components results in improved maintainability and testability of code and increases the comprehensibility, reliability, and quality of software. By the integration concept, large software applications are developed by reusing these modules. Existing software components can communicate using common interfaces and are assembled in a complementary way into complex

systems. This approach has the advantage of limited development effort and shorter development times.

The need to transfer software development practices to the environmental modelling community is self–evident and widely recognised (e.g. McIntosh et al., 2005; Rizzoli et al., 2008; Scheller et al., 2010; Verweij et al., 2010; Holzworth et al., 2010). Just like with any piece of software, the development of models becomes more efficient when code, or process components, can be reused. Component reuse has become even more urgent because environmental research increasingly addresses interactions in large coupled systems. The development of models of such systems requires collaboration between model builders of various disciplines and the integration of their associated domains such as hydrology, ecology, or economics. The construction of such integrated models calls for approaches adopting the principles from component–based software development. In modelling, the concept of interoperability implies that process components are constructed of limited complexity, while the concept of integration implies construction of integrated models by coupling process components of limited complexity (e.g. Argent, 2004; Hinkel, 2009).

These ideas to support code reuse have partly been implemented in environmental modelling. Design and implementation of process components is supported by a broad range of tools (e.g. Sklar, 2007; ArcGIS, 2013; R Development Core Team, 2013; MATLAB, 2013; ExtendSim, 2013), including domain specific languages (e.g. Wesseling et al., 1996; Pullar, 2004; Wang and Pullar, 2005; Degenne et al., 2009) and environmental modelling frameworks (e.g. Hill et al., 2004; Moore and Tindall, 2005; Argent et al., 2009; Peckham et al., 2013; David et al., 2013). These tools ease the development and coupling of generic process components by providing data types and specific functionality tailored to the development of environmental models. Even with these tools, however, it remains time consuming to program and couple limited complexity process components. Moreover, it requires mostly specialist programming knowledge to do so, while it would often be preferable if domain specialists can implement the models themselves (Karssenberg et al., 2007). We think three issues still limit wider application of component based modelling in the environmental sciences.

One is that the languages used in modelling frameworks still rely on application programming interfaces of their underlying programming languages. This raises a problem when components need to be coupled from different modelling frameworks, because code needs to be written to bridge the gap between different programming languages. We need to tear down this technical interoperability barrier by developing common interfaces to a wide range of types of component models.

The second issue is due to the heterogeneity of environmental science as a discipline. Most modelling frameworks evolved from specific domains such as hydrology, ecology, or engineering, which has resulted in a heterogeneous set of modelling frameworks, each tailored to modelling paradigms best applicable to a particular discipline. This resulted in modelling frameworks and component models that are largely incompatible, because they are different regarding the representation of processes, discretisation of space and time, or syntax of the modelling language used.

Third, common programming languages used for the modelling frameworks use low–level application programming interfaces for coupling process components. This implies that inputs and outputs of process components need to be described in terms of data types or functions used by the low–level programming languages. As a result, model builders are not able to express in the process component interface the scientific meaning and processes embedded in the process components. The semantics are not present in the interfaces of the process components, and thus cannot be used to transfer information between process components or to evaluate the validity of couplings between process components. This is a major limitation, because in order to prevent mistakes it is equally essential to validate couplings between components based on semantics of these components, as it is to do type checking in programming languages based upon the calculation executed by the functions, a principle which is paramount in type–safe programming languages.

Fortunately, we are not on new ground here. Widely used approaches exist that deal with formalising semantics, and these have found their application in environmental sciences. Application examples can be found in the semantic annotation of environmental data (e.g. Fonseca et al., 2002; Saiful Islam and Piasecki, 2006; Madin et al., 2007; Rasinmäki et al., 2009), formalising activities (e.g. Kuhn, 2001) or data retrieval (e.g. Baglioni et al., 2008; Lutz and Klien, 2006), in the integration of web–based or service–oriented architectures (e.g Athanasis et al., 2005; Best et al., 2007), or in ontology–aided simulations (e.g. Beck et al., 2010; Janssen et al., 2011). Semantic knowledge is also used to describe component interfaces (e.g. Rizzoli et al., 2008; Athanasiadis et al., 2011) and to extend modelling frameworks (e.g. Villa, 2007; Villa et al., 2009). However, these approaches only address individual tasks in the model development cycle. A software framework appropriate for domain specialists, which semantically supports the tasks of process component construction, composition, and execution, is still missing.

We propose a formalisation supporting the semantic annotation of process–based spatio–temporal components, and their coupling to integrated models. First, we define the requirements of a semantic model construction process and determine the essential building blocks needed for the development of integrated models. For these building blocks, we describe the required information to allow seamless coupling, and provide a means to formalise this information. We then derive an implementation independent representation from this conceptual formalisation and present a software architecture able to evaluate model building blocks and compositions based on the semantic descriptions. To provide a tool suitable for domain scientists we show how an existing declarative framework for integrated modelling (see Chapter 2) can be extended to generate semantically annotated models. Finally, we discuss the design choices and semantic enrichment of the model building blocks in the context of model execution and model assessment and outline further applications that assist the model builder with the development of reusable model building blocks.

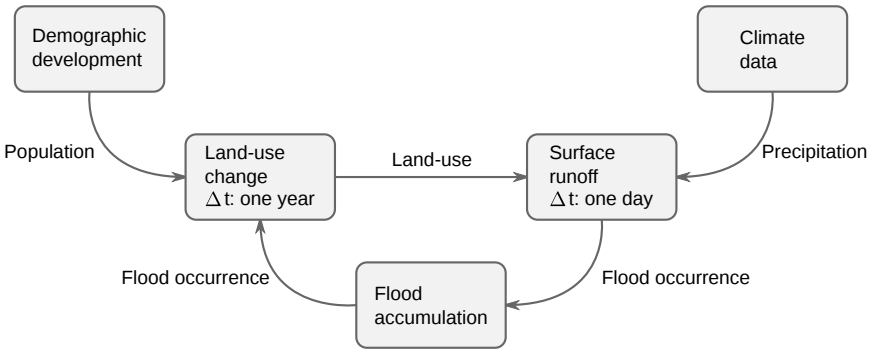

Figure 6.1 Example of an integrated model for flood risk assessment composed of modular building blocks. Land–use change and surface runoff are process components describing spatio–temporal phenomena. Demographic and climate data providers supply data sets. The flood accumulation block adjusts for the different time steps of the process components. The arrows display the direction of the exchanged variables.

6.2 Design criteria for knowledge integration into a modelling framework

Software frameworks support modellers in the construction of integrated models by providing tailored functionality and building blocks for component construction and coupling. However, a framework cannot provide intrinsic support for all potential model application cases. The construction of integrated models requires a coupling of various components from different domains such as hydrology or economics, each domain with its individual conventions. Component models can operate at individual spatial or temporal discretisations, and process representations in the components can be formalised according to different modelling paradigms. Although research on unifying methodologies, for example the field–based and object–based modelling paradigms, is ongoing (e.g. Frank, 2001; Grenon and Smith, 2004; Kjenstad, 2006; Goodchild et al., 2007; Couclelis, 2010), a modelling framework with universal support for all modelling purposes is not foreseeable. As a result, we focus on the development of a framework supporting a modeller in the construction of process–based spatio–temporal models. Despite the constraint to this particular type of models, a broad range of applications is possible (e.g. van der Knijff et al., 2010; Hiemstra et al., 2011; van Beek et al., 2011).

A domain specialist such as a hydrologist could be interested, for example, in the construction of an integrated model suitable to assess flood risk in a catchment (see Figure 6.1). In this case, flooding determines land–use change as flooding will reduce the probability of certain land–use types in the future. In addition, land–use has a feedback on flooding as it affects runoff rates and thus hydrology and flooding occurrence. Feedback mechanisms are represented by building blocks with domain specific functionality. The first building block simulates land–use change such as increased surface sealing driven by an increasing demand for urban areas. The second building block simulates

hydrological processes such as surface runoff. The feedback between the domains is simulated by the exchange of the following variables: land–use is provided as input to the hydrological process component, and runoff is returned to the land–use change model. We assume a yearly time step in the land–use change component and a daily time step in the hydrological component. Due to the discrepancy in the temporal discretisation between the components, the hydrological output variable needs to be aggregated on a yearly interval using an accumulator building block.

A modeller can use the process component as building block to program processes representing real world phenomena such as surface runoff. These processes are preferably constructed with spatial data types and domain specific operations corresponding to domain concepts (Karssenberg, 2002), like raster–based maps and flow operations such as the kinematic wave. A modeller also needs to integrate knowledge outside his own profession, such as using components modelling land–use change or calculating economic damage. Following software engineering practices (e.g. Gamma et al., 1995; Szyperski, 2002; Booch et al., 2007), the process representations need to be encapsulated within process components. The inputs and outputs of a process component need to be exposed by defined interfaces, comparable to interface definition languages like CORBA (2012) or modelling notations like UML or SysML (e.g. Booch et al., 2005; Friedenthal et al., 2008).

Reusable process components contain the scientific process descriptions but should not depend on specific application cases by incorporating data sets such as catchment maps. By separating process knowledge and data, process components can be easily used in different case studies (e.g. Villa, 2007; Voinov and Cerco, 2010). A data–provider building block is needed supplying input data such as demographic or meteorologic data.

Modellers have to deal with different spatial and temporal discretisations when coupling of process components and data providers. For example, runoff processes are represented with a fine temporal discretisation of days or hours whereas land–use change processes typically use larger time steps such as years. A modeller therefore needs to describe scaling operations for an appropriate coupling, for example, an aggregation of the daily discharge values to a total value for a year. However, process components and data providers are only reusable if there is no need to modify the internal component code when coupling to other model building blocks. To allow for the coupling of components with different spatial or temporal discretisations, we introduce an accumulator building block (see, e.g. Chapter 2). The accumulator obtains a set of output variables from process components or data providers, and a modeller can describe operations for spatial and temporal scaling operations within the accumulator.

Modellers need a great degree of flexibility at different levels to build integrated models such as the one shown in Figure 6.1. A modelling framework needs to support this (see Figure 6.2). Process components can be developed with existing environmental modelling languages (e.g. Wesseling et al., 1996; Pullar, 2004), which provide flexibility in process component implementation (Karssenberg, 2002). In hydrology, for example, a modeller can straightforward describe environmental process with operations based on the Map Algebra concept (e.g. Tomlin, 1990; Pullar, 2001; Karssenberg et al., 2007). Less flexibility exists in coupling process components, because a universal Model Algebra, i.e.

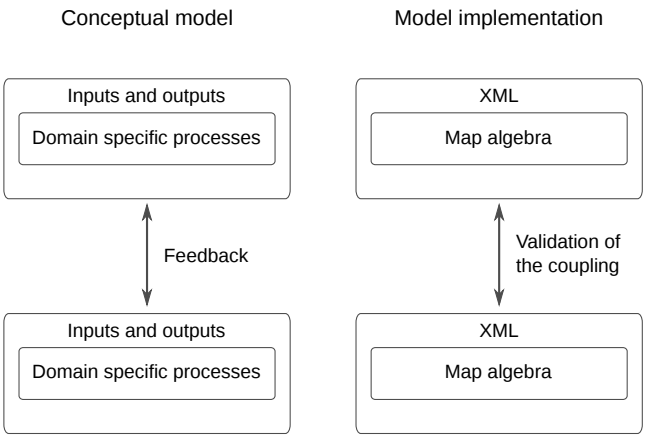

Figure 6.2 Building blocks providing flexibility in the construction process of coupled models in the conceptual model phase (left) and its equivalent in the implementation with the help of a modelling framework (right).

a straightforward algebra for coupling process components, does not yet exist. A first step towards such a Model Algebra is to formalise the information about characteristics of the model building blocks, with a focus on characteristics required for coupling building blocks. In Section 6.3, we standardise interfaces and exchanged variables, and develop a formalisation expressed in natural language describing characteristics for process components, data providers, and accumulators. This formalisation needs to include a description of inputs and outputs of the building blocks, and preferably a description of the environmental processes represented by the building block (Polhill and Gotts, 2009).

The formalisation defined in Section 6.3 describes characteristics of model building blocks in a natural language. A computer readable format is required such that a modelling framework can process the information. In Section 6.4, we develop an XML representation based on the formalised building blocks. The XML only describes model building blocks but does not provide an evaluation of a, by the modeller, proposed coupling. To enable this, Section 6.4 presents an evaluation scheme that evaluates couplings between building blocks based upon their XML description.

The XML format introduced in Section 6.4 allows a formal description of the model building blocks and can be used to evaluate couplings. However, XML is rather a low–level technical format while a modeller is describing processes in the model building blocks with domain concepts. We therefore show in Section 6.5 an approach to derive the XML representation from model scripts implemented with Python (2013) and Map Algebra constructs.

6.3 Formalisation of model building blocks

Because model builders are both involved in component construction and coupling, both stages of the modelling process need to be semantically supported. We first describe the

functionalities of the three different building blocks introduced in the previous section, and then derive a uniform formal representation that can be used for all building blocks.

6.3.1 Types of model building blocks

Process components: building blocks for process representation

We here define the *process component* as the building block to represent dynamic behaviour of a subsystem. Process components calculate individual processes like, for example, infiltration or sediment deposition. A process component is restricted to the representation of a well–defined process. Therefore, it is assumed to have a constant time step duration Δt for all transient calculations represented by the process component. It also typically uses the same spatial discretisation for all parameters and variables used in the component. A process component simulates a specific process over a set of discrete time steps t with duration Δt according to (Beck et al., 1993; van Deursen, 1995):

$$Z_t = \begin{cases} Z_0 & \text{for } t = 0 \\ f(Z_{t-1}, I_t, P) & \text{otherwise} \end{cases} \tag{6.1}$$

Here, f is the state transition function representing the process simulated by the process component. This function is constructed and implemented by the model builder and can be composed of, for example, differential equations, update rules or probabilistic rule sets (Karssenberg and de Jong, 2005a). I_t refers to the input values of the current time step t which are obtained from other building blocks. P refers to the parameter values that are piecewise constant during the simulation time. Feedback is modelled by incorporating the state variables of the previous time step Z_{t-1} in the calculation of the current state variables Z_t. Note that most variables in Equation 6.1 are spatial.

A variety of data types can be used to represent the attribute properties of parameters, state variables and input values. For example, intrinsic data types provided by common programming languages such as integer or double can be used for the representation of lumped values. Additionally, higher level constructs such as arrays, lists or user defined data types can be used. Additional information on attribute properties such as spatial discretisation and measurement units need to be provided as well.

Several software packages dedicated to environmental modelling exist (e.g. Wesseling et al., 1996; Pullar, 2001; Schwanghart and Kuhn, 2010) that can be used by modellers to implement the state transition function f and thus a process component. These packages provide functions that originate from libraries providing support for spatial data types (e.g. GDAL Development Team, 2013; OGR, 2013) or spatio–temporal modelling and analysis (e.g. Chakhar and Mousseau, 2007; Mennis, 2010). Here, we neglect the implementation details of process components and instead focus on formalising the functional characteristics that are relevant to ensure generic interoperability with other model building blocks.

Data providers: building blocks providing environmental data sets

We define the *data provider* as the building block providing data sets from available sources such as files or databases. Data providers can be connected to process compo-

nents that need to be provided with static or temporal data sets. An example of a static data set is the initial state Z_0 in Equation 6.1 defining the initial groundwater storage in a catchment for a process component modelling groundwater flow. Temporal data sets are, for example, time series of maps containing precipitation values, or randomly perturbed time series for scenario simulations. Temporal data sets can be used to provide the input I_t of Equation 6.1 to the process components. This means the specification of the data set provided is not dependent on the runtime of a model. The information about spatial extent and temporal characteristics such as the time step duration and temporal extent are determined at the model coupling stage.

Accumulators: building blocks bridging spatio–temporal discrepancies
We define the *accumulator* as the building block used to bridge discrepancies in space and time discretisation or attribute types between two building blocks. Accumulators allow for the coupling of the building blocks with different temporal or spatial characteristics without the need to modify the individual building blocks. Temporal discrepancies occur for example when coupling a component with a daily time step to a component with a monthly time step. Spatial discrepancies are for example different grid cell sizes.

The calculation of the output O of an accumulator is done by a temporal aggregation of variables according to:

$$O = f(I_{t_i}, \ldots, I_{t_j}) \tag{6.2}$$

Here, the time steps I_{t_i} and I_{t_j} describe an interval in which the accumulator obtains variables such as all daily variables of the last month. The function f is either merging the input variables to an individual variable such as in calculating the arithmetic mean, or a filter operation selecting individual variables such as the maximum operation. If only one input variable I_t is provided as input to the accumulator, f is performing a conversion operation on the individual input variable. In this case, f adapts variables, for example, by a spatial resampling operation, data type or measurement unit conversions.

6.3.2 Formalising model building blocks

Process components, data providers, and accumulators are the building blocks required to construct an integrated model. While the internal working of the building blocks is different, their external presentation with respect to interoperability is similar: each building block holds an interface, and exposes temporal and type information about the exchanged variables. We therefore develop a uniform description of a generic model building block able to express each of the specialisations for the process components, data providers, and accumulators.

Using a single description for all building blocks comes with several benefits. First, a uniform description allows a seamless replacement of building blocks. For example, a data provider can be replaced by a process component. Second, compositions of building blocks, i.e. an integrated model, can be represented using the same uniform description also used for the building blocks themselves. Third, a uniform description eases interoperability with software using the model such as execution or analysis tools. Individual building blocks and coupled models can be used in the same manner.

We define a uniform formalisation of a model building block by the tuple:

```
buildingBlock = (interface, invariants, mutables)
```

That is, each of the process components, data providers and accumulators is a building block, and has generic elements formalising the interface, the invariants and mutables. The elements are described below and specialised for the types of building blocks.

Interface
Entailing a common interface enables a building block to obtain input I_t (Equation 6.1) from other building blocks, and to provide its own state Z_t as an output. A process component can require and provide a number of inputs and outputs, the interface is therefore defined by a list of interface ports:

```
interface = ([interfacePort])
```

with an interface port specified as

```
interfacePort = (direction, variable)
```

Each interface port has an `incoming` or `outgoing` direction element indicating that a variable is required as input or provided as output.

For process components, the interface holds both incoming and outgoing ports. The interface of the data provider in general only holds interface ports with an outgoing direction. Accumulators bridge discrepancies between building blocks, their interface holds input and output ports.

Variable This element describes the variable that is provided or required at an interface port. A variable can hold for example lumped values such as measurement values obtained at a discharge gauge, or spatially distributed data sets such as temperature fields or groundwater recharge. To allow for a generic representation of a variable, we formalise the description by a set of constraints:

```
variable = (timeConstraints, unitType, variableType,
            variableConstraints, extentConstraints)
```

The first element describes the temporal characteristics of a building block by defining a temporal horizon given by a lower `Tmin` and upper `Tmax` time step boundary. The time step duration Δt that can be represented by a building block is also constrained by lower and upper boundaries:

```
timeConstraints = (Tmin, Tmax, dTmin, dTmax)
```

For a data provider, the `timeConstraints` are known at construction time. Therefore, the lower and upper time steps of the `timeConstraints` elements are required, and the length of the time step is described by coinciding `dTmin` and `dTmax`. A data provider can

thereby provide, for example, a 30–year precipitation data set on a daily basis starting from the year 1960.

For a process component, the elements `Tmin` and `Tmax` describing the temporal horizon are empty as implemented processes are independent of a specific start and end time, for example in calculations describing runoff processes. With the constraints of the time step Δt, a modeller can specify the characteristic time step of a process. A chosen implementation in a process component may be valid within a certain range of time step durations, while outside this range the process calculations become incorrect. For example, a particular process description of surface runoff, e.g. a kinematic wave, could be used for time step durations between one minute and one hour, but not for a time step of one year.

For the accumulators, the `Tmin` and `Tmax` elements are empty. On the accumulator input and output interface, coinciding `dTmin` and `dTmax` describe the time step duration of the providing and receiving building blocks, respectively.

The `unitType` and `variableConstraints` elements are always defined for all the building blocks. The `unitType` element describes the measurement unit of an environmental variable:

```
unitType = (dimension, unit)
```

With the first element, the dimension of a measurement, such as length or mass, of a variable is described. The second element describes the specific measure of the dimension such as metre or kilogramme. The range of allowed values of a variable can be limited by lower and upper boundaries:

```
variableConstraints = (min, max)
```

The `variableType` element describes the data type of a variable. Variables can be described by intrinsic data types of generic programming languages like `bool` or `float`, or described by data types provided by libraries for spatial modelling. The spatial characteristics of a variable are described by a set of constraints:

```
extentConstraints = (lengthConstraints, rowConstraints,
                     colConstraints, locationConstraints)
```

The elements describe the allowed ranges of discretisation specifications such as the minimum and maximum values for the length of a grid cell. For a process component, the allowed grid cell size suitable for the implemented process needs to be specified. For a data provider, the ranges of each element brake down to individual values indicating the exact geographical extent and location of a variable.

Invariants and mutables

The interface formalisation enables a straightforward coupling of a building block. To express information in an environmental modelling context, we define two more elements within a building block:

```
invariants = ([parameter], [initialState])
```

and

```
mutables = ([stateVariable])
```

Here, the parameter, initialState and stateVariable elements are lists of the variables described above. The invariants describe elements that are constant at run time. Following Equation 6.1, the invariants are given by the parameter P and the initial state Z_0 of a process component. The state variable Z_t changes at runtime. Describing these elements enables, for example, calibration tools to identify which parameters can be accessed at component initialisation, or data assimilation tools to modify state variables at runtime.

6.3.3 Coupling model building blocks

To develop a coupled system, interactions between the building blocks need to be specified considering the following conditions.

First, defining the interactions between the building blocks should be possible without modification of their internal working. Modifying building blocks to enable coupling should be avoided, as it is against the principle of independent, generic, building blocks that can be reused in other models. When this condition is satisfied, a flexible coupling of different building blocks can be realised. Second, the interaction between building blocks is realised by an exchange of environmental variables between the building blocks such as a variable holding precipitation data. Third, a description of a coupling needs to contain the necessary information such that consistency checks can be performed and a sound model setup can be guaranteed.

We define the logical connection of two interface ports as a link between two building blocks. Demanding that the direction element of the first interface port is outgoing and of the second interface port is incoming, the source and the target of the link are specified as follows:

```
link = (interfacePort1, interfacePort2)
```

The specification of the link is stated in one direction. A bidirectional interaction to model feedback processes between two process components can be realised by the specification of two separate links with opposite directions.

6.4 The formalisation implemented in a software prototype

We now show how the formalisation introduced in the previous section is applied to the development phases. We first explain how a machine–oriented, implementation–independent format represents the proposed formalisation, which is illustrated with the help of the example model shown in Figure 6.1. In addition, we describe how this format can be used to validate proposed couplings between building blocks, which is required to guarantee a sound model setup.

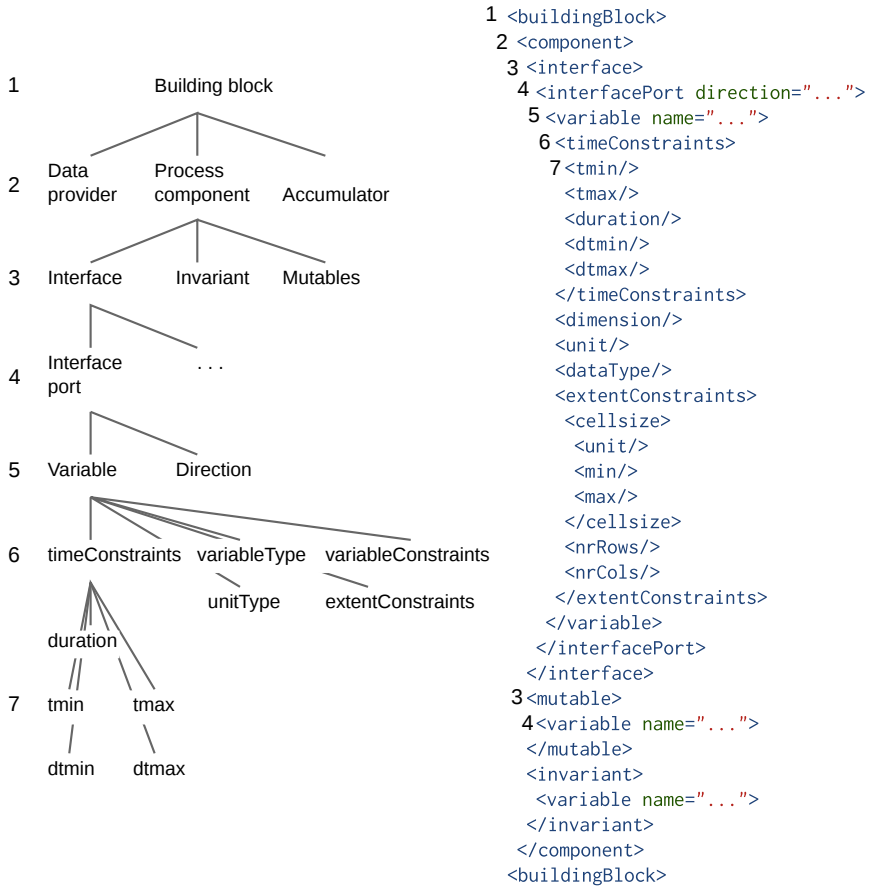

Figure 6.3 Building block formalisation and its representation in a hierarchical XML structure. Elements of the formalisation shown on the left are mapped to XML elements (content of the elements not shown). Numbers refer to the hierarchical level of the formalisation and the corresponding depth in the XML tree structure.

6.4.1 Semantic enrichment of model building blocks

The natural language notation of the formalisation in Section 6.3 is given independently of a specific modelling tool or software implementation. We use the Extensible Markup Language (XML, 2008) as a machine–oriented format to represent our formalisation. XML allows us to express data and semantics in a hierarchical structure in one document, and automatic validation with XML Schema documents, or transformation by XSLT to other geoscientific formats such as GML (2013), WaterML (2013) or UncertML (Williams et al., 2009). The usage of XML as a qualified base format to separately describe model implementation and knowledge elements has previously been shown (e.g. Rizzoli et al., 2008; Villa et al., 2009; Athanasiadis et al., 2011). XML is also used as a foundation for formalisms used by the semantic web community like the Resource Description Framework (RDF, 2004) or the Web Ontology Language (OWL, 2004).

The formalisation introduced in Section 6.3 is hierarchically composed by *is–a* and *has–a* relationships. For example, a process component is a building block, and an inter-

```
<component name="SurfaceRunoff">
 <interface>
  <interfacePort direction="incoming">
   <variable name="precipitation">
    <timeConstraints>
     <tmin/>
     <tmax/>
     <duration>seconds</duration>
     <dtmin>1</dtmin>
     <dtmax>86400</dtmax>
    </timeConstraints>
    <dimension>mass</dimension>
    <unit>mm</unit>
    <dataType>Scalar</dataType>
    <!-- omitted -->
   </variable>
  </interfacePort>
  <interfacePort direction="incoming">
   <variable name="landuse">
    <extentConstraints>
     <cellsize>
      <unit>metres</unit>
      <min>50</min>
      <max>250</max>
     </cellsize>
     <nrRows/>
     <nrCols/>
    </extentConstraints>
    <!-- omitted -->
   </variable>
  </interfacePort>
 </interface>
</component>
```

```
<interfacePort direction="outgoing">
 <variable name="precipitation">
  <timeConstraints>
   <tmin/>1990-01-01T00:00:00Z<tmin/>
   <tmax/>2010-01-01T00:00:00Z<tmax/>
   <duration>seconds</duration>
   <dtmin>3600</dtmin>
   <dtmax>3600</dtmax>
  </timeConstraints>
  <dimension>mass</dimension>
  <unit>mm</unit>
  <dataType>Scalar</dataType>
  ...
```

```
<interfacePort direction="outgoing">
 <variable name="landuse">
  <extentConstraints>
   <cellsize>
    <unit>metres</unit>
    <min>100</min>
    <max>1000</max>
   </cellsize>
   <nrRows/>
   <nrCols/>
  </extentConstraints>
  ...
```

Figure 6.4 Illustration of XML segments from model building blocks introduced in the example model. On the left, the interface description of the surface runoff process component with the temporal constraints for the incoming precipitation and spatial constraints for the incoming land–use is shown. On the upper right, the output interface of the climate data provider with exact values is shown. The lower right side shows the output interface of the land–use change process component.

face port has a direction element. The left side of Figure 6.3 is a graphical representation of the elements of the formalisation and their hierarchical relationships. The root node on the top is the most generic description of a model building block, which is specialised in different levels further on. The levels shown in Figure 6.3 correspond to the levels given in Section 6.3.

The structure of the conceptual formalisation is directly mapped into a tree structure in the XML representation. Each element of the formalisation corresponds to an XML element given, for example, by the `<interface>` tag. XML elements can hold child elements and it is therefore possible to express the hierarchical structure of the formalisation. The leaf nodes of the XML such as the `<min>` element can be filled with content describing the characteristics of a building block.

Figure 6.4 shows an example of such XML elements filled with content. The left–hand side of the figure shows the XML document corresponding to the process component modelling surface runoff. For brevity, we omit the output port providing the surface

runoff, and only give an example for the temporal and spatial constraints specified by the modeller by means of the two component inputs. The information specified in the time constraints section state that the process implementation of the component is always valid, and that an appropriate time step is within a range of one second and one day. The spatial characteristics of the required land–use input are shown with the constraints for the cell size. The required cell size of the land cover lies between within 50 and 250 metres. The process implementation of the surface runoff component is not limited to a certain number of rows or columns, respectively.

On the right–hand side of Figure 6.4, we show the description of the building blocks coupled to the surface runoff component. The XML excerpt for the data provider supplying the precipitation data is shown on the top. The XML structure of the interface port from the data provider equals the interface port description of the surface runoff component. As the information for the variables from the data provider is determined at model composition time, the elements are completed with fixed values. In our example, a 20–year time series, starting from 1990 and with an hourly time step, is specified by the ISO 8601 format in the `timeConstraints` section.

The second required input for the surface runoff component is provided as an output from the land–use change process component. The corresponding XML segment is shown in the lower right side of Figure 6.4. As the providing land–use change component is a process component as well, the variable of the output port is also specified by constraints. Here, the cell size of the output variable is within the range of 100 and 1000 metres.

6.4.2 Validation of model compositions

The semantic information of the building block interfaces can be used to evaluate the correctness of model setups given by modellers. In addition to the general type checking offered by generic programming languages, the semantic content of interfaces and exchanged variables can be incorporated into the evaluation process. A semantic modelling framework can exceed an evaluation based on intrinsic data types such as `float` by including user–specified information, and prevent a modeller to execute ill–defined model setups as, for example, in coupling process components with different measurement units.

The validation algorithm needs to be capable of dealing with variables constrained by value ranges. That is, two process components with given ranges are linkable if actual values fall in the intersection set of the two allowed ranges specified in the interface ports. In the example of Figure 6.5A, we assume a permitted cell size for the process component PC_1 between 75 and 100 m, and for PC_2 between 75 and 750 m. A coupling of the process components is valid only if the actual cell size is in the range of 75 and 100 metres. A validity of the links between building blocks is determined by the actual values that are imposed by an application case, such as the discretisation for a certain catchment. The data providers D_1 and D_2 supply actual values and activate the evaluation of a model setup according to the algorithm shown in Table 6.1.

The evaluation of a model setup is thereby done in several stages. Foremost, all links in the coupled model are assessed. That is, the existence of the specified variables at

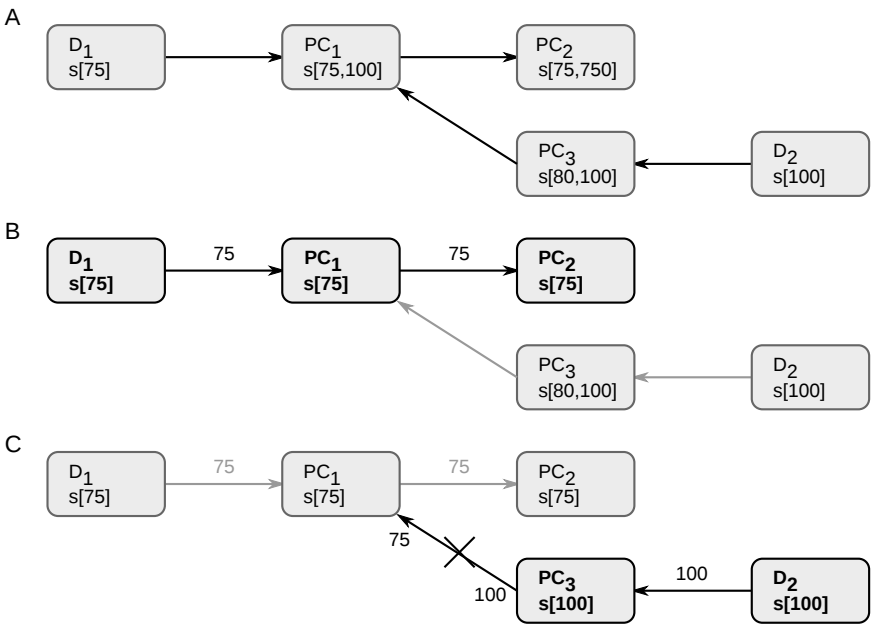

Figure 6.5 Propagating actual values in constraint–based models. A) Initial model setup before validation of the linkages. B) After validation of the links between D_1, PC_1 and PC_2. C) After validation of the whole model setup. Model building blocks hold constraints for a variable, here the cell size s. D_i denote data providers, and PC_i process components.

Table 6.1 Simplified algorithm for the evaluation of model setups with constrained building blocks. [v] and [l,u] denote restrictions to actual values or constraints to lower and upper boundary values, respectively.

```
for all links
  validate interface ports
step A
  for all links between data providers D and process components PC
    if D[v] is in range PC[l,u]
      PC[l,u] = D[v]
   else link not valid
step B
  for all links between determined PC and PC
    if PC[v] is in range PC[l,u]
      PC[l,u] = D[v]
    else link not valid
step C
  for all links between determined PC
    if PC1[v] != PC2[v]
      link is not valid
```

the interface ports and the direction of the ports are evaluated. Afterwards, three steps are performed. First, the data providers are determined as they supply the actual values and impose these on the coupled process components. Figure 6.5B and step A of the algorithm show the flow of information imposed from the data provider D_1. The actual value from D_1 lies within the permitted range of the process component PC_1. A coupling of D_1 and PC_1 is valid, and therefore a cell size of 75 m is imposed on PC_1.

In the step A, all coupled data providers and process components are evaluated, and the characteristics of the coupled process components are determined. Afterwards, the continuing propagation of values is evaluated. This procedure is required to determine the characteristics for process components that do not receive input from a data provider. This case is shown in step B of the algorithm and applies to the process component PC_2 in Figure 6.5B. The determined value of the providing process component is evaluated against the permitted range of the receiving process component and, if applicable, the characteristics of PC_2 are set.

In the final phase, couplings between process components with determined characteristics are evaluated (step C). Figure 6.5C shows the case that two process components are coupled to data providers, here PC_3 is supplied with a 100 m cell size from a data provider D_2. The specified coupling between PC_2 and PC_3 is not valid as the imposed actual values of both process components are not in agreement. In this case, a modeller needs to interconnect an accumulator with an appropriate scaling operation.

In the explanatory example above, we only use the cell size element of the formalisation to clarify the information flow and evaluation steps. For a valid model setup, all elements of the exchanged variables connected to the interface ports need to be evaluated.

6.5 A model building language to generate the formal representation

The XML representation of the model building block formalisation shown in the previous section is given independently of a specific programming language or modelling environment. However, a manual preparation of model semantics embedded in XML documents remains a tedious task for environmental scientists. We therefore extend a modelling framework (Karssenberg et al., 2007; Chapter 2) with a semantic framework able to generate and process the XML formalisation. With this prototype, a modeller can build components and integrated models in a descriptive way without explicit writing of XML documents.

The existing modelling framework uses the general purpose scripting language Python (2013) as descriptive language for model construction. The framework provides templates for model building blocks and operations for spatial modelling. We extend the architecture with a layer automatically generating the proposed XML formalisation for the building blocks and integrated models.

This approach is illustrated with the generation of a building block XML document as shown in the left side of Figure 6.4 with the help of a script showing the implementation

Table 6.2 Python code for a model building block. Shown is the surface runoff process component described by the XML fragment in Figure 6.4. Process components consist of standard Python constructs, methods and data types from libraries for environmental modelling, and sections to specify value constraints.

```
1 class SurfaceRunoff(object):
2   """ Hortonian runoff calculation
3
4       Incoming:
5         -precipitation
6           time: dTmin=1, dTmax=1440
7           unit: dimension=mass, unit=mm
8           constraints: min=0.0
9
10        -landuse
11        -dem
12
13      Outgoing:
14        runoff
15  """
16  def __init__(self, dem):
17    self.ldd = lddcreate(dem, 1E35, 1E35, 1E35, 1E35)
18    self.landuse = nominal(0)
19    self.precipitation = scalar(0)
20    self.runoff = scalar(0)
21
22  def process(self):
23    infiltrationCapacity = lookupscalar("infCap.txt", self.landuse)
24    self.runoff = accuthresholdflux(self.ldd, self.precipitation, ↩
          infiltrationCapacity)
```

of the surface runoff process component. Table 6.2 shows the corresponding model script as specified by a modeller. The model scripts use the object–oriented features of the Python programming language supporting a component–based development style, and operations for spatial modelling (Wesseling et al., 1996).

The surface runoff component is initialised in the `init` section and the process calculated per time step is described in the `process` section. The spatial variables representing the input and output of the process component are set to an initial value of zero (lines 18–20), and a local drain direction map is calculated based on the digital elevation map. For each time step (lines 23 and 24), a map with infiltration capacity values is calculated based on the incoming land–use classes. The infiltration capacity and precipitation are used to calculate the surface runoff by the flow operator `accuthresholdflux`.

The constraints of the variables are given in the comment section of the Python model class (lines 2–14). For each variable and element of the formalisation, a modeller can

specify allowed ranges. The modelling framework parses the section and extracts the boundary values. Information such as data types of variables or actual grid discretisation for data providers are derived by introspection at initialisation of the model building blocks. The automatically derived and user specified characteristics are then combined to an XML document.

6.5.1 Further usage of the formalisation

Until now, we outlined the application of the formalisation in the stages of component construction and coupling to enhance interoperability and to guarantee a proper model setup. More application cases building upon the formalisation will provide benefits to the model builder. One application case is to use the standardised building block description to generate documentation in human–readable format. For example, transforming model descriptions to input files of processing tools such as Graphviz (Gansner and North, 2000) can be used to provide a visual representation of the constructed model, or automatically generated HTML pages can provide detailed information about requirements and outputs of model building blocks. This kind of component documentation can provide modellers an easy entry point in understanding and integration of building blocks created by other modellers.

Another application case is to use the formalisation in the stage of model execution. While executing a model, the modelling framework can combine runtime information with the building block specification into a combined document holding for example time stepping of each of the variables of a coupled model in combination with their full type information. Additionally, meta information about the model setup such as date and wall clock time of the model run, and parameter values used by calibration schemes can be integrated. A combined document can thereby hold relevant information to identify a model run, and therefore serve as modelling logbook recommended for reproducible research and good modelling practices (e.g. Refsgaard and Henriksen, 2004; Scholten et al., 2007; Schmolke et al., 2010). A document holding runtime and component characteristics can also be used in the model assessment stage. An evaluation or visualisation tool can use the generated document to query, for example with XQuery (e.g. Rasinmäki, 2009), to retrieve necessary information for their internal processing about time stepping or data type information of requested variables.

6.6 Discussion and conclusion

We developed a formalisation of model building blocks suitable for the construction of integrated models, supporting the tasks of component construction and coupling for discrete spatio–temporal modelling. It was shown that three basic types of building blocks, i.e. process components, data providers, and accumulators, can be represented by a single formalisation. We illustrated the concepts with a prototype implementation in XML, and its application in the evaluation of model setups. For user convenience, we illustrated how a model building environment can be used to automatically generate the formalisation and therefore relieve the modeller from manually generating the XML files.

The described formalisation is in our opinion an important step towards increased flexibility in the construction process of integrated models, because it provides standardisation of components and their couplings, and thus supports reuse of code and process knowledge embedded in this code. This is mainly due to two design choices. One is that we apply component–based software engineering techniques in the environmental modelling domain. The proposed formalisation promotes a development of independent model building blocks that are described by a common interface, and tailored to the construction of process–based environmental models. Secondly, in the design of our formalisation, the starting point was a uniform formalisation applicable to each of the three building blocks, which was shown to be possible. While the internal function of the process components, data providers, and accumulators differ, their interfaces regarding interaction with other building blocks are equal. As a result, tools that use our formalisation do not need to differentiate between building blocks which greatly simplifies concepts and implementation. This unification of building blocks is thus far not widely applied in the environmental modelling domain. Important contributions (e.g. Saiful Islam and Piasecki, 2006; Rizzoli et al., 2008; Athanasiadis et al., 2011; Granell et al., 2013a) use a similar approach providing formalised descriptions of integrated models. However, these studies use different descriptions for models and data, making an additional step to use both approaches in one modelling environment necessary. By defining the variables of the process component building blocks in a generic way with value constraints, we can additionally describe the other building blocks required (data providers and accumulators) using the same formal description. Using the same formal representation is beneficial as it allows a seamless exchange of building blocks of the same type or a replacement with building blocks of other types.

The need to enrich building blocks with semantic information may hamper wide application of the approach, because the information needs to be provided by the model builder. While XML is one of the preferred formats for machine representation, practical concerns in the preparation of the XML documents arise for a modeller. A manual generation of XML documents provides the greatest flexibility and expressiveness, see for example the LIQUID modelling framework (Branger et al., 2010) or the management of forest resource data (Rasinmäki et al., 2009). However, a manual generation of XML documents requires a continuous synchronisation of implementation and semantic description, and editing XML is tedious. An opposite approach is to use existing process component construction tools to automatically generate a formal representation of process components built with these tools. For example, domain specific languages such as PCRCalc (Wesseling et al., 1996) or NetLogo (Sklar, 2007), or modelling languages such as SysML could be extended to generate XML documents. This approach provides a more tailored solution for modellers but is limited to the functionality provided by the toolbox. Our approach of extending a generic scripting language with environmental modelling capabilities and a semantic framework provides an intermediate environment with full flexibility for the development of model building blocks and a convenient interface for the modeller.

We propose to use the Python documentation strings as location for the specification of semantics and constraints of model building blocks and variables. Hence, XML docu-

ments can be automatically generated from the Python code. Other options to provide a modeller a means to specify additional information are possible. First, conventional methods provided by a modelling framework such as `setUnit` can be used throughout the model code. Second, decorators such as `@unit` can be used to annotate methods or classes with additional semantics. Decorators are available in programming languages such as Java or C# and used, for example, in the OMS (e.g. David et al., 2013). Both, decorators and framework methods can be utilised to generate the building block formalisation. However, framework methods are specific to a modelling framework and they will be distributed over the source code making model code less readable and portable. Decorators provide a clear location to specify semantics but are still bound to framework functionality. With our approach to add additional information in the comment section of the Python class (see Table 6.2), we use a centralised place for semantic descriptions of the building blocks making the model code easier readable and understandable. As comments do not interfere in script execution, model descriptions are portable and can in general be executed with or without the semantic annotation framework. The intrusiveness of semantic framework requirements on component descriptions is therefore minimised (see Lloyd et al., 2011). Additionally, tools already using the documentation string sections such as Sphinx (2013) can be customised to incorporate the semantics given by the modeller.

Our proposed formalisation is tailored to the construction of spatio–temporal models. We support two data types for the environmental variables, standard arithmetic data types provided by the Python language, and raster–based data types and operations from the PCRaster library. With these data types and operations, modellers can construct a wide range of models for domains such as hydrology or ecology. However, modellers might need additional data models such as user–defined data types, or data models for feature data. Due to the hierarchical setup of the formalisation and resulting XML, it is possible to accommodate different data models by providing a formalisation of the new data model and inserting the new description at the level of the current variable description. Nevertheless, it is required that the new variable is again formalised in a hierarchical data model such that the modelling framework can evaluate the new variable type by recursively traversing and comparing the additional XML structure. Potential first candidates for extension can be the HDF data type (HDF5, 2010) as a widely used and metadata augmented hierarchical data storage, and the climate and forecast metadata conventions (NetCDF-CF, 2013).

A second but not less interesting subject concerns the extensibility of the formalisation towards other modelling paradigms such as integrating multiple instances of process components behaving as individuals or agents. Extending the framework into this direction requires significant efforts, as currently, for example, no location changes are considered and therefore moving process components cannot be represented. Also, the functioning of the links need to be extended to cover functions or relationships to calculate for example distances between model agents or to express family relationships, respectively. However, research on integrating and formalising field–based and agent–based modelling paradigms (e.g. Kjenstad, 2006; Bian, 2007; Goodchild et al., 2007) can be used as a starting point.

The formalisation and prototype implementation so far provides a basis for a construction of model building blocks and integrated models. Further developments will be directed towards the formalisation of modular components for model execution and evaluation schemes such as modules for sensitivity analysis or uncertainty estimation, and integration of these modules into the modelling environment.

7 Synthesis

Environmental models are valuable tools for the understanding of natural processes and way to manage our living environment. Their application is essential in evaluating and forecasting human–natural systems. The construction and assessment of computer models representing these systems, commonly referred to as integrated modelling, is difficult as environmental scientists face both conceptual and technical challenges. Conceptually, modellers need to identify the process description of each spatio–temporal sub–system, they need to define appropriate scaling operations to realise a sound multi–scale coupling between sub–systems, and they need to select observational data and to develop methods to evaluate the uncertainties in their model results. Technically, model builders need to program component models by implementing transition functions that represent environmental processes, specify data exchange to formalise the interactions between component models, and execute the model for analysis purposes.

The main objective of this dissertation was to develop a component–based modelling framework providing domain specialists with one modelling environment for the construction and stochastic analysis of multi–scale integrated models. The focus was on the sound execution of integrated models, bridging space and time discrepancies, establishing uncertainty analysis and enhancing the semantic interoperability of component models. We will now discuss these aspects from the technical as well as the model builder's perspective.

7.1 Execution of multi–scale integrated models

To obtain reliable results from integrated models, there is an evident need for a precise execution order of component models and information exchange between component models. Two chapters of this dissertation evaluate different methodologies to execute either component models or integrated models. In Chapter 2, we developed a modelling framework that uses a centralised client–server approach to realise the execution of integrated models. Using the time steps in the component models and couplings specified by the model developer, the framework scheduler generates a shared time line, determines the order of component execution, and performs the data transfer at the most appropriate moments. In Chapter 3, we propose a conceptually different approach that does not use a framework scheduler. Instead, we use a request–reply approach that considers component models as blocks that request data from other components to realise the execution of integrated models. Here, the order and execution of component models is initiated by the input requests specified by the model builder as part of the state transition function.

Both modelling frameworks provide modellers with generic model building blocks for the implementation of the state transition function of component models and for the exchange of information between component models, including transfer of the spatio–

temporal resolution using accumulators. The frameworks build on the existing map algebra concepts (e.g. Tomlin, 1990; Karssenberg et al., 2007) for the implementation of spatio–temporal processes in component models and aggregation operations within the accumulators. The presented modelling frameworks organise the execution and data exchange in integrated models as follows.

At first, the frameworks enable us to simulate the dynamic behaviour of environmental systems by progressing individual components as required for the execution of single models, or as execution per component model in an integrated setup. The client–server approach introduced in Chapter 2 uses a central instance (the server) with overall knowledge to initiate and execute each individual time step of all component models and accumulators (the clients). The component models in the request–reply approach presented in Chapter 3, on the contrary, are able to execute several time steps autonomously when progressing until data is requested by other component models at a certain time step.

Secondly, a modelling framework needs to determine a temporal order to execute component models in a multi–scale coupling. This is a common issue in environmental modelling which can be addressed in different ways. The client–server approach introduced here generates a shared time line between all component models, and uses this to determine the execution order based on the current time steps of the component models. By using a request–reply approach, the execution order is implicitly predefined by the input requirement of the component models given by the model builder.

Thirdly, interactions and therefore data exchanges between component models need to be established. In the client–server approach, the management layer of the modelling framework initiates all data transfer. This means, the server obtains outputs variables from component models and distributes these as inputs among the component models. In the request–reply approach, the data exchange occurs between directly coupled component models, and the transfer is initiated by the requesting component model as part of the state transition function.

Finally, the modelling frameworks need to be able to provide support for post–processing and model output analysis. We therefore extended the request–reply mechanism to Monte Carlo simulations. This is done by using component–based development approaches and separating the component development and execution mechanisms. This allows for a straightforward extension of further execution mechanisms by potential reuse of already implemented execution mechanisms as shown for the `DynamicFramework` in the `MonteCarloFramework` shown in the request–reply approach. While support for Monte Carlo simulation is not included in the current model execution management layer of the client–server framework, no technical or conceptual barriers impede the inclusion of such execution mechanisms.

The choice of execution methodology has implications for the software implementation of the modelling framework and the execution management. For the client–server approach, the component models can be implemented in a light–weighted way. In general, the component models only use information on how to proceed to the next time step, independently from the existence of other component models in the integrated model. Obtaining data, the progress over simulation time, and execution in individual model

runs or Monte Carlo simulations is organised by the central instance of the modelling framework. The implementation complexity for the framework developer is therefore on the server side, where the shared time line and an automated schedule generation, execution, data transfer and storage according to different execution methodologies need to be implemented. The request–reply mechanism requires less effort to implement, as the execution order is passed on to the modeller. The overall knowledge about all participating component models and accumulators on the server side, however, offers higher optimisation potential in the client–server approach. A centralised instance can potentially determine the execution schedule in advance, allowing for runtime optimisation as in concurrent execution of component models as done, for example, in Barthel et al. (2008), or a less extensive data storage as required for the accumulators.

A centralised management also offers the potential to integrate the increasing amount of functionality provided by online–services such as web–based data infrastructures (e.g. Huang et al., 2011; Ames et al., 2012) or spatio–temporal aggregation services (e.g. Stasch et al., 2012). Best et al. (2007), for example, concatenate web services within a scientific workflow to obtain, process and visualise environmental data. However, such workflows rarely incorporate dynamic behaviour and feedback effects between web–services. Extending the current data transfer mechanism of the client–server framework with a network–based protocol would allow access and drive independent web–services and lead to a to dynamic, web–based and distributed modelling architecture (see, e.g. Bastin et al., 2013).

In combination with the two different model execution approaches, we also provided the modeller with two different approaches to specify the input requirements and thus the couplings between component models. In Chapter 2, we introduced a model algebra as additional instrument to specify how components interact and which variables need to be exchanged. However, the coupling of component models can be specified by a modeller in any order independent of the actual execution order. The scheduler of the modelling framework will determine this without being transparent for the model builder. The function–based notation used in the request–reply approach of Chapter 3 allows a modeller to specify the input requirements as part of the state transition function. This approach is to be preferred for use by modeller builders as it resembles the traditional function–based development of state transition functions and gives modellers more control over the execution flow.

7.2 Bridging spatial and temporal discretisation differences

Spatial and temporal misalignment is a common problem when constructing integrated models. Component–based model development allows for the construction of component models at particular spatial and temporal scales but prohibits the modification of process implementations to fulfil requirements of a particular coupling. In this dissertation, we use the accumulator as a programmable general–purpose model building block that executes user–defined scaling operations at model runtime. To allow for an appropriate scaling, a modelling framework needs to determine the correct time intervals

at model runtime, and to obtain the corresponding output variables. This will give modellers access to these variables and allow them to express spatial and temporal scaling operations.

Both modelling frameworks and presented execution approaches, the client–server approach in Chapter 2 and the request–reply approach used in Chapter 3 can be used to schedule and execute accumulators. The client–server approach requires the scheduling algorithm to consider the execution order of the component, the providing component must be executed before the requesting component such that variables are accessible for an accumulator. In the request–reply approach used in Chapter 3, the accumulator triggers the exertion of the providing component and no prerequisites for a correct execution order of the coupled component models is required.

Both frameworks provide modellers an instrument to access and traverse intervals of output variables obtained from the providing components. With that, modellers can use common map algebra operations to implement scaling operations in space and time. Compared to software frameworks using system programming languages such as OpenMI (e.g. Moore and Tindall, 2005) or ESMF (e.g. Hill et al., 2004), modellers can express spatio–temporal scaling operations on the same data types and with the same operations that are used for the construction and coupling of components as well.

Still, the accumulators only provide a technical solution and require attention from modellers when implementing scaling operations. This includes, for example, an appropriate choice of scaling algorithms, the specification of an order of spatial and temporal scaling, and the assessment of error propagation resulting from aggregation.

7.3 Integration of legacy code

In Chapter 2, 3 and 4, the environmental processes described in the component models were newly implemented with map algebra operations provided in the modelling framework. A common situation, however, is the integration of existing model applications into coupled systems. Integration of such "external" model applications is to be preferred when the desired functionality cannot be reimplemented easily within a modelling framework. By wrapping the external model applications as shown in this research, it is technically possible to reuse the functionality in component models of the modelling framework. A further integration of those wrapped applications into coupled models, however, most likely requires scaling operations. The required scaling and potentially other required conversions such as in unit conversion can be done by modeller by means of the accumulators as demonstrated for the coupling of the land use change model *Ruimtemodel* (e.g. Engelen et al., 2011) to the hydrological model WetSpa (e.g. Wang et al., 1996) shown in Chapter 4.

In this dissertation, we used two wrapping approaches. The first wrapping approach shown in Chapter 5 uses the Application Programming Interface (API) of the PCRaster modelling environment to integrate the external MODFLOW groundwater model (Harbaugh et al., 2000). The approach described allows software engineers to integrate a particular external model application, and allows them to develop operations that con-

form to the notation of the modelling environment, as shown in the example scripts of Chapter 5. When access to the source code of the external application is available, runtime optimisation such as memory–based exchange of data could be realised instead of transferring inputs and outputs via disk storage. The effort to implementation the API approach, however, can be significant. It can be justified, for example, when the external model application will be used regularly within the new modelling framework.

The second wrapping approach (Chapter 4) calls the application within a component model and is more straightforward to implement for an environmental modeller but can be less efficient due to overhead needed for file transfer via disk storage and potential format conversions for the input and output data. This wrapping approach depends on the expertise of modellers concerning data formats and software specifics of the external model application.

Both wrapping approaches enable modellers to construct new, reusable component models from existing functionality. The API approach provides a common notation and offers a potential for runtime optimisation but is hard to implement for environmental modellers. They may therefore favour integrating a model application by using a simple system call for its execution, giving them a rapid way to integrate external knowledge into their own models. If the functionality of the external model emerges as a useful addition for a modelling framework this can guide software engineers to develop implementations with higher runtime performance.

7.4 Semantic interoperability

Chapter 6 evaluates how the scientific meaning of numerical implementations can be expressed and used by an environmental modeller. We propose an implementation–independent formalisation describing the semantics of integrated models. We define component models, accumulators, and data providers as main building blocks for the construction of integrated models and formalise invariant characteristics required to describe the spatial and temporal context of these building blocks and their interactions.

An implementation–independent storage and representation of scientific knowledge can technically be realised with the XML format. Both framework implementations that were discussed in Chapter 2 and Chapter 3 can be extended such that the formal description can be generated automatically by the modelling frameworks to describe semantic content of component models and their couplings. In addition to the type–safe evaluation of the intrinsic data types of programming languages, the frameworks can evaluate couplings of component models based on the semantics of domain–specific data types to prevent the exchange of mismatching information between component models.

For an environmental modeller, semantics are important because they capture the scientific meaning and contents of component models in a more transparent way than their pure numerical implementation. An improved comprehensibility of component models can stimulate their reuse and alleviate their integration into coupled models. This

formalisation can be further used to extend the documentation of component models and enhance the archiving of model components.

However, adding additional semantic information about component models especially formalised in XML format can be tedious. Model builders can be expected to prefer specifying the semantic information in the same language as used in the component models. We showed that with a few functional extensions the Python language and modelling framework can be used to add the semantic contents.

We formalised the key functionalities of a modelling framework tho ease the development of process–based spatio–temporal models. This is certainly only a particular subset of methodologies or data types used in the construction of environmental models. Although the approach is simplified, however, it includes the construction, coupling and evaluation of component–based models with the formalisation applicable in a single modelling environment. An integrated modelling framework with embedded formalisation and more straightforward application of semantics could lead to a higher acceptance by the environmental modelling community.

7.5 Future perspectives

We developed and evaluated approaches that can be incorporated into environmental modelling frameworks and enhance the construction, reuse and interoperability of environmental component models. With software engineering methodologies such as the component–based software development and a standardised construction process in one modelling environment, further applications and extensions of the modelling framework are feasible. These are now briefly elaborated.

7.5.1 Assessment of integrated models

Although the scientific findings of component models with individual environmental process descriptions are well–known, the combination of different component models into integrated systems still offers a potential for gaining new scientific insights or to determine the uncertainty source in integrated models. The modelling frameworks presented here support a plug–and–play construction of component models and couplings of these components. The frameworks can assist environmental scientists with the assessment of integrated models in a more straightforward way, and allow quantification of the propagation of uncertainty. A more flexible construction of a model setup facilitates the evaluation and comparison of different model structures in terms of the sensitivity to changes in model structure. By using and exchanging different components modelling the same environmental process, the performance of different process implementations can be assessed. Fenicia et al. (2008), for example, evaluated two different model structures for a simple rainfall–runoff model, and afterwards proposed a flexible framework to assess alternative model structures (Fenicia et al., 2011; Kavetski and Fenicia, 2011). A further execution mechanism could be added to the modelling framework generating a large number of candidate models from available classes of component models, and consequently a number of different model structures (e.g. Marshall

et al., 2006). An automated assessment of these candidate models can lead to a more appropriate identification of the model structure of complex distributed hydrological models. In addition, by exchanging accumulators, the influence of scaling algorithms or the sequence of spatial and temporal scaling operations can be evaluated. Executing component models either individually or as part of a coupled model, as achievable with the two execution frameworks in Chapter 3, can help to identify if errors are located in particular component models or in the integration. The evaluation of different process implementations offers also potential for model simplification. Replacing component models with complex process implementations with simpler components while retaining the process characteristics, for example, can help to reduce the model runtimes.

The scientific assessment of integrated models remains a challenge despite the easier technical model development by using the modelling framework and component–based development practices. So far, only a few studies quantify the effects involved in the coupling of multi–scale integrated models (e.g. Claessens et al., 2009; Moreira et al., 2009; Elag et al., 2011), and there are no analyses to date that quantify uncertainty across modelling components that incorporate regional climate, energy and agricultural systems, socio–economics and technology (Hibbard and Janetos, 2013). An appropriate assessment of integrated models, however, depends on data availability, system characteristics, and modellers' opinion (Hsu et al., 1999).

Integrated models holding several environmental processes require modellers, for example, to determine appropriate assessment metrics. The reduction of complex models to a set of basic emergent behaviours, such as a catchment model that is validated using discharge values obtained at one outlet, might not reflect all spatial and temporal characteristics. It is therefore recommended to include multi–scale and multi–site criteria in the model assessment procedure, such as applying several catchments of different sizes to identify scale–dependent parameters (Krysanova et al., 2007). In addition, multiple statistics can be combined to multi–criteria validation (Krysanova et al., 2007; Bellocchi et al., 2010), such as using discharge data and remotely sensed soil moisture data to calibrate hydrological models (e.g. Sutanudjaja et al., 2014).

Ideally, the validation is performed as an on–going process during the development of an integrated model (van Oijen, 2002). Still, a formal procedure for the assessment of integrated models is inexistent. Extending existing guidelines for good modelling practices (e.g. Risbey et al., 2005; Jakeman et al., 2006; Matott et al., 2009) to include characteristics of integrated models, such as the selection of appropriate spatial scales and time steps (e.g. Beven, 2004; Wainwright and Mulligan, 2004), considering the influence of (dis)aggregation algorithms used in the coupling, and multi–criteria validation (e.g. Moriasi et al., 2007), remains a task for the environmental modelling community.

7.5.2 Enhancing the model lifecycle

The widespread reuse of component models by scientists still remains a challenge although good software engineering practices and good modelling practices emphasise the benefits for the field of environmental modelling. A lack of scientific credibility of component models can be a reason that currently hampers reuse and application of

existing component models. A profound testing procedure of component models as part of quality assurance could provide a remedy, and appropriate testing strategies should be included as part of the environmental model development cycle (see, e.g. Scheller et al., 2010). In software engineering, dedicated testing frameworks for automated repeated testing of software units and their integration into larger systems exist for every major system programming language. By incorporating functionality required for environmental modelling—such as spatial data types—into existing unit testing frameworks these could be valuable additions to integrated modelling frameworks. Automated testing in the model analysis phase can ensure that the model performance is met continuously or assumptions remain valid while parts of the model are redesigned or exchanged by other component models.

7.5.3 Extending the accumulators

Finally, additional research is needed to examine the role of the accumulator concept and to improve the notation of aggregating or disaggregating operations.

The component models implemented in this dissertation follow the field–based modelling paradigm, and establishing interactions between component models raised the problem to resolve spatial and temporal discrepancies. Using the accumulators to bridge this discrepancies offers a solution for a particular modelling paradigm available for the construction of environmental models. The individual–based modelling paradigm, however, is widely used to model human–natural systems as well and applied, for example, to model ecological systems (e.g. Grimm et al., 2006; Perez and Dragicevic, 2012) or to model human decisions (e.g. Becu et al., 2003; Li, 2012). Further research can evaluate the usage of accumulators to bridge different modelling paradigms and to integrate models following the individual–based modelling paradigm. Primarily, modellers can use the accumulators to implement appropriate conversions between component models with individual–based and field–based process implementations. Another alternative is to extend the framework and accumulators such that a modeller can access different component models in a way comparable to the iteration over time intervals.

The modelling frameworks and the accumulators applied in this research use two–dimensional raster–based data types. However, the concept aggregation and disaggregation within accumulators can be extended to other data types as well. An obvious extension is to incorporate three–dimensional voxel–based data types and scaling operations thereon. Developing and testing a set of accumulators that execute operations on individual time steps, on a specific time window as required for example to incorporate a time lag in a component model, on time intervals or on all past time steps could be useful for the development of a generic set of time operations resembling the classification of map algebra operations. A combination of these findings could eventually lead to a ready for use, multi–dimensional algebra for integrated modelling.

Bibliography

ABBOTT, M. B., J. C. BATHURST, J. A. CUNGE, P. E. O'CONNELL and J. RASMUSSEN (1986a), An introduction to the European Hydrological System - Système Hydrologique Européen, "SHE", 1: History and philosophy of a physically-based, distributed modelling system. Journal of Hydrology 87(1–2), 45–59.

ABBOTT, M. B., J. C. BATHURST, J. A. CUNGE, P. E. O'CONNELL and J. RASMUSSEN (1986b), An introduction to the European Hydrological System - Système Hydrologique Européen, "SHE", 2: Structure of a physically-based, distributed modelling system. Journal of Hydrology 87(1–2), 61–77.

ABDELLA, Y. and K. ALFREDSEN (2010), A GIS toolset for automated processing and analysis of radar precipitation data. Computers & Geosciences 36(4), 422–429.

AHRENDS, H., M. MAST, C. RODGERS and H. KUNSTMANN (2008), Coupled hydrological-economic modelling for optimised irrigated cultivation in a semi-arid catchment of West Africa. Environmental Modelling & Software 23(4), 385–395.

AMES, D. P., J. S. HORSBURGH, Y. CAO, J. KADLEC, T. WHITEAKER and D. VALENTINE (2012), HydroDesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis. Environmental Modelling & Software 37, 146–156.

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., A. VOINOV, T. MAXWELL, S. M. CUDDY, J. M. RAHMAN, S. SEATON, R. A. VERTESSY and R. D. BRADDOCK (2006), Comparing modelling frameworks - A workshop approach. Environmental Modelling & Software 21(7), 895–910.

ARGENT, R. M., J.-M. PERRAUD, J. M. RAHMAN, R. B. GRAYSON and G. M. PODGER (2009), A new approach to water quality modelling and environmental decision support systems. Environmental Modelling & Software 24(7), 809–818.

ARMSTRONG, C. W., R. W. FORD and G. D. RILEY (2009), Coupling integrated Earth System Model components with BFG2. Concurrency and Computation: Practice and Experience 21(6), 767–791.

ATHANASIADIS, I. N., A.-E. RIZZOLI, M. DONATELLI and L. CARLINI (2011), Enriching environmental software model interfaces through ontology-based tools. International Journal of Applied Systemic Studies 4(1–2), 94–105.

ATHANASIS, N., K. KALABOKIDIS, M. VAITIS and N. SOULAKELLIS (2005), The Emerge of Semantic Geoportals. In: On the Move to Meaningful Internet Systems 2005: OTM Workshops, Springer Berlin / Heidelberg, 1127–1136.

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.

BAGLIONI, M., E. GIOVANNETTI, M. V. MASSEROTTI, C. RENSO and L. SPINSANTI (2008), Ontology-supported Querying of Geographical Databases. Transactions in GIS 12(Supplement 1), 31–44.

BARNES, B., K. MOKANY and M. RODERICK (2007), Allocation within a generic scaling framework. Ecological Modelling 201(2), 223–232.

BARREDO, J. I., M. KASANKO, N. MCCORMICK and C. LAVALLE (2003), Modelling dynamic spatial processes: Simulation of urban future scenarios through cellular automata. Landscape and Urban Planning 64(3), 145–160.

BARREDO, J. I., L. DEMICHELI, C. LAVALLE, M. KASANKO and N. MCCORMICK (2004), Modelling future urban scenarios in developing countries: An application case study in Lagos, Nigeria. Environment and Planning B: Planning and Design 31(1), 65–84.

BARTHEL, R., S. JANISCH, N. SCHWARZ, A. TRIFKOVIC, D. NICKEL, C. SCHULZ and W. MAUSER (2008), An integrated modelling framework for simulating regional-scale actor responses to global change in the water domain. Environmental Modelling & Software 23(9), 1095–1121.

BASTIN, L., D. CORNFORD, R. JONES, G. B. M. HEUVELINK, E. PEBESMA, C. STASCH, S. NATIVI, P. MAZZETTI and M. WILLIAMS (2013), Managing uncertainty in integrated environmental modelling: The UncertWeb framework. Environmental Modelling and Software 39, 116–134.

BATELAAN, O. and F. DE SMEDT (2004), Seepage, a new Modflow Drain package. Ground Water 42(4), 576–588.

BATELAAN, O., Z.-M. WANG and F. DE SMEDT (1996), An adaptive GIS toolbox for hydrological modelling. In: K. Kovar and H.-P. Nachtnebel, eds., Application of Geographic Information Systems in Hydrology and Water Resources Management, vol. 235, IAHS, 3–9.

BECK, H., K. MORGAN, Y. JUNG, S. GRUNWALD, H. KWON and J. WU (2010), Ontology-based simulation in agricultural systems modeling. Agricultural Systems 103(7), 463–477.

BECK, K. and C. ANDRES (2004), Extreme Programming Explained: Embracing Change. Addison-Wesley, 2nd edn.

BECK, M. B., A. J. JAKEMAN and M. J. MCALEER (1993), Construction and evaluation of models of environmental systems. In: M. B. Beck, A. J. Jakeman and M. J. McAleer, eds., Modelling change in environmental systems, New York: John Wiley & Sons Ltd., 3–35.

BECKER, B. P. and H. SCHÜTTRUMPF (2011), An OpenMI module for the groundwater flow simulation programme Feflow. Journal of Hydroinformatics 13(1), 1–12.

BECU, N., P. PEREZ, A. WALKER, O. BARRETEAU and C. L. PAGE (2003), Agent based simulation of a small catchment water management in northern Thailand: Description of the CATCHSCAPE model. Ecological Modelling 170(2–3), 319–331.

BEHNEL, S., R. BRADSHAW, C. CITRO, L. DALCIN, D. S. SELJEBOTN and K. SMITH (2011), Cython: The best of both worlds. Computing in Science and Engineering 13(2), 31–39.

BELLOCCHI, G., M. RIVINGTON, M. DONATELLI and K. MATTHEWS (2010), Validation of biophysical models: Issues and methodologies. A review. Agronomy for Sustainable Development 30(1), 109–130.

BENENSON, I. and P. TORRENS (2004), Geosimulation: Automata-based modeling of urban phenomena. Chichester: Wiley.

BEST, B. D., P. N. HALPIN, E. FUJIOKA, A. J. READ, S. S. QIAN, L. J. HAZEN and R. S. SCHICK (2007), Geospatial web services within a scientific workflow: Predicting marine mammal habitats in a dynamic environment. Ecological Informatics 2(3), 210–223.

BEVEN, K. and A. BINLEY (1992), The future of distributed models: model calibration and uncertainty prediction. Hydrological Processes 6(3), 279–298.

BEVEN, K. J. (2004), Rainfall-Runoff Modelling: The Primer. Wiley.

BEYDEDA, S., M. BOOK and V. GRUHN, eds. (2005), Model-Driven Software Development. Springer.

BIAN, L. (2007), Object-oriented representation of environmental phenomena: Is everything best represented as an object? Annals of the Association of American Geographers 97(2), 267–281.

BIERKENS, M. F. P., P. A. FINKE and P. DE WILLIGEN (2000), Upscaling and downscaling methods for environmental research. Kluwer.

BLÖSCHL, G. and M. SIVAPALAN (1995), Scale issues in hydrological modelling: a review. Hydrological Processes 9(3–4), 251–290.

BOEHM, B. W. (1988), SPIRAL MODEL OF SOFTWARE DEVELOPMENT AND ENHANCEMENT. Computer 21(5), 61–72.

BOLTE, J. P., D. W. HULSE, S. V. GREGORY and C. SMITH (2007), Modeling biocomplexity - actors, landscapes and alternative futures. Environmental Modelling & Software 22(5), 570–579.

BOOCH, G., J. RUMBAUGH and I. JACOBSON (2005), Unified Modeling Language User Guide. Addison-Wesley, 2nd edn.

BOOCH, G., R. A. MAKSIMCHUK, M. W. ENGEL, B. J. YOUNG, J. CONALLEN and K. A. HOUSTON (2007), Object Oriented Analysis and Design with Applications. Addison Wesley.

BOOST.PYTHON (2013), Boost C++ libraries. URL http://www.boost.org/libs/python/.

BRANGER, F., I. BRAUD, S. DEBIONNE, P. VIALLET, J. DEHOTIN, H. HENINE, Y. NEDELEC and S. ANQUETIN (2010), Towards multi-scale integrated hydrological models using the LIQUID® framework. Overview of the concepts and first application examples. Environmental Modelling & Software 25(12), 1672–1681.

BROEKX, S., I. LIEKENS, W. PEELAERTS, L. DE NOCKER, D. LANDUYT, J. STAES, P. MEIRE, M. SCHAAFSMA, W. VAN REETH, O. VAN DEN KERCKHOVE and T. CERULUS (2013), A web application to support the quantification and valuation of ecosystem services. Environmental Impact Assessment Review 40, 65–74.

BROOKE, A., D. KENDRICK, A. MEERAUS and R. RAMAN (1998), GAMS, a User's Guide. GAMS Development Corporation.

BROOKS, F. P., JR (1987), No Silver Bullet Essence and Accidents of Software Engineering. Computer 20(4), 10–19.

BUCCELLA, A., A. CECHICH and P. FILLOTTRANI (2009), Ontology-driven geographic information integration: A survey of current approaches. Computers & Geosciences 35(4), 710–723.

BULATEWICZ, T., X. YANG, J. M. PETERSON, S. STAGGENBORG, S. M. WELCH and D. R. STEWARD (2010), Accessible integration of agriculture, groundwater, and economic models using the Open Modeling Interface (OpenMI): methodology and initial results. Hydrology and Earth System Sciences 14(3), 521–534.

BULATEWICZ, T., A. ALLEN, J. M. PETERSON, S. STAGGENBORG, S. M. WELCH and D. R. STEWARD (2013), The Simple Script Wrapper for OpenMI: Enabling interdisciplinary modeling studies. Environmental Modelling & Software 39, 283–294.

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

BURROUGH, P. A. and R. A. MCDONNELL (1998), Principles of Geographical Information Systems. Oxford University Press, 2nd edn.

BURROUGH, P. A., D. KARSSENBERG and W. P. A. VAN DEURSEN (2005), Environmental Modelling with PCRaster. In: D. J. Maguire, M. F. Goodchild and M. Batty, eds., GIS, Spatial Analysis and Modeling, Redlands, California: ESRI, 333–356.

CARRERA-HERNÁNDEZ, J. J. and S. J. GASKIN (2006), The groundwater modeling tool for GRASS (GMTG): Open source groundwater flow modeling. Computers & Geosciences 32(3), 339–351.

CASTRONOVA, A. M. and J. L. GOODALL (2010), A generic approach for developing process-level hydrologic modeling components. Environmental Modelling & Software 25(7), 819–825.

CERCO, C. F., D. TILLMAN and J. D. HAGY (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., G. CÂMARA, U. MOURA DE FREITAS and F. ALMEIDA (2009), Yet Another Map Algebra. GeoInformatica 13(2), 183–202.

CHAKHAR, S. and V. MOUSSEAU (2007), An algebra for multicriteria spatial modeling. Computers, Environment and Urban Systems 31(5), 572–596.

CHOW, V. T., D. R. MAIDMENT and L. W. MAYS (1988), Applied Hydrology. New York: McGraw-Hill.

CLAESSENS, L., J. M. SCHOORL, P. H. VERBURG, L. GERAEDTS and A. VELDKAMP (2009), Modelling interactions and feedback mechanisms between land use change and landscape processes. Agriculture, Ecosystems & Environment 129(1–3), 157–170.

COHEN, D., M. LINDVALL and P. COSTA (2004), An Introduction to Agile Methods. Advances in Computers 62(C), 1–66.

COLLINS, N., G. THEURICH, C. DELUCA, M. SUAREZ, A. TRAYANOV, V. BALAJI, P. LI, W. YANG, C. HILL and A. DA SILVA (2005), Design and implementation of components in the Earth System Modeling Framework. International Journal of High Performance Computing Applications 19(3), 341–350.

CORBA (2012), Common Object Request Broker Architecture Specification Version 3.3. Object Management Group. URL http://www.omg.org/spec/CORBA/3.3.

COUCLELIS, H. (2010), Ontologies of geographic information. International Journal of Geographical Information Science 24(12), 1785–1809.

CSDMS (2013), Community Surface Dynamics Modeling System website. URL http://csdms.colorado.edu/.

DAKOS, V., E. H. VAN NES, R. DONANGELO, H. FORT and M. SCHEFFER (2009), Spatial correlation as leading indicator of catastrophic shifts. Theoretical Ecology 3(3), 163–174.

DAMS, J., E. SALVADORE, T. V. DAELE, V. NTEGEKA, P. WILLEMS and O. BATELAAN (2012), Spatio-temporal impact of climate change on the groundwater system. Hydrology and Earth System Sciences 16(5), 1517–1531.

DAMS, J., J. DUJARDIN, R. REGGERS, I. BASHIR, F. CANTERS and O. BATELAAN (2013), Mapping impervious surface change from remote sensing for hydrological modeling. Journal of Hydrology 485, 84–95.

DAVID, O., J. C. ASCOUGH II, W. LLOYD, T. R. GREEN, K. W. ROJAS, G. H. LEAVESLEY and L. R. AHUJA (2013), A software engineering perspective on environmental modeling framework design: The Object Modeling System. Environmental Modelling & Software 39, 201–213.

DE KOK, J.-L. and M. GROSSMANN (2010), Large-scale assessment of flood risk and the effects of mitigation measures along the Elbe River. Natural Hazards 52(1), 143–166.

DE KOK, J.-L., S. KOFALK, J. BERLEKAMP, B. HAHN and H. WIND (2009), From Design to Application of a Decision-support System for Integrated River-basin Management. Water Resources Management 23(9), 1781–1811.

DE KOK, J.-L., G. ENGELEN and J. MAES (2010), Towards model component reuse for the design of simulation models - a case study for ICZM. In: D. A. Swayne, W. Yang, A. A. Voinov, A. Rizzoli and T. Filatova, eds., Modelling for Environment's Sake, vol. 2 of *Proceedings of the iEMSs Fifth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2010)*, International Environmental Modelling and Software Society, Ottawa, Canada, 1215–1222.

DE KOK, J.-L., L. POELMANS, G. ENGELEN, I. ULJEE and L. VAN ESCH (2012), Spatial-dynamic visualization of long-term scenarios for demographic, social-economic and environmental change in Flanders. In: R. Seppelt, A. A. Voinov, S. Lange and D. Bankamp, eds., Managing Resources of a Limited Planet: Pathways and Visions under Uncertainty, Proceedings of the iEMSs Sixth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2012), International Environmental Modelling and Software Society, Leipzig, Germany, 1984–1991.

DE ROO, A., J. THIELEN, P. SALAMON, K. BOGNER, S. NOBERT, H. CLOKE, D. DEMERITT, J. YOUNIS, M. KALAS, K. BÓDIS, D. MURARO and F. PAPPENBERGER (2011), Quality control, validation and user feedback of the European Flood Alert System (EFAS). International Journal of Digital Earth 4(SUPPL. 1), 77–90.

DE SMEDT, F., Y. B. LIU and S. GEBREMESKEL (2000), Hydrologic modeling on a catchment scale using GIS and remote sensed land use information. In: C. A. Brebbia, ed., Risk analysis II, WIT press, Southampton, Boston, 295–304.

DE VASCONCELOS, M. J. P., A. GONCALVES, F. X. CATRY, J. U. PAUL and F. BARROS (2002), A working prototype of a dynamic geographical information system. International Journal of Geographical Information Science 16(1), 69–91.

DEGENNE, P., D. LO SEEN, D. PARIGOT, R. FORAX, A. TRAN, A. AIT LAHCEN, O. CURÉ and R. JEANSOULIN (2009), Design of a Domain Specific Language for modelling processes in landscapes. Ecological Modelling 220(24), 3527–3535.

DOHERTY, J. and J. M. JOHNSTON (2003), Methodologies for calibration and predictive analysis of a watershed model. Journal of the American Water Resources Association 39(2), 251–265.

DONCHYTS, G. and B. JAGERS (2010), DeltaShell - An open modelling environment. In: D. A. Swayne, W. Yang, A. A. Voinov, A. Rizzoli and T. Filatova, eds., Modelling for Environment's Sake, vol. 2 of *Proceedings of the iEMSs Fifth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2010)*, International Environmental Modelling and Software Society, Ottawa, Canada, 1050–1057.

DONCHYTS, G., S. HUMMEL, S. VANEÇEK, J. GROOS, A. HARPER, R. KNAPEN, J. GREGERSEN, P. SCHADE, A. ANTONELLO and P. GIJSBERS (2010), OpenMI 2.0 - What's new? In: D. A. Swayne, W. Yang, A. A. Voinov, A. Rizzoli and T. Filatova, eds., Modelling for Environment's Sake, vol. 2 of *Proceedings of the iEMSs Fifth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2010)*, International Environmental Modelling and Software Society, Ottawa, Canada, 1174–1181.

DOUCET, A., S. GODSILL and C. ANDRIEU (2000), On sequential Monte Carlo sampling methods for Bayesian filtering. Statistics and Computing 10(3), 197–208.

DOUCET, A., N. DE FREITAS and N. GORDON (2001), Sequential Monte Carlo Methods in Practice. Statistics for Engineering and Information Science, New York: Springer.

ELAG, M. and J. L. GOODALL (2013), An ontology for component-based models of water resource systems. Water Resources Research 49(8), 5077–5091.

ELAG, M. M., J. L. GOODALL and A. M. CASTRONOVA (2011), Feedback loops and temporal misalignment in component-based hydrologic modeling. Water Resources Research 47(12), W12520.

ENGELEN, G., A. HAGEN-ZANKER, A. C. M. DE NIJS, A. MAAS, J. VAN LOON, B. STRAATMAN, R. WHITE, I. ULJEE, M. VAN DER MEULEN and J. HURKENS (2005), Kalibratie en validatie van de LeefOmgevingsVerkenner (Calibration and validation of the LeefOmgevingsVerkenner). In Dutch, URL http://www.pbl.nl/publicaties/2005/KalibratieEnValidatieVanDeLeefOmgevingsVerkenner.

ENGELEN, G., C. LAVALLE, J. I. BARREDO, M. VAN DER MEULEN and R. WHITE (2007), The Moland Modelling Framework for Urban and Regional Land-Use Dynamics. In: E. Koomen, J. Stillwell, A. Bakema and H. J. Scholten, eds., Modelling Land-Use Change, vol. 90 of *The GeoJournal Library*, Springer Netherlands, 297–320.

ENGELEN, G., L. POELMANS, I. ULJEE, J.-L. DE KOK and L. VAN ESCH (2011), De Vlaamse Ruimte in 4 Wereldbeelden - Scenarioverkenning 2050 - Kwantitatieve Verwerking (The Flemish Space in 4 World Views - Scenario Exploration 2050 - Quantification). A study for the Policy Research Center for Spatial Planning and Housing. VITO Report 2011/RMA/R/363. Flemish Institute for Technological Research (VITO), Mol, Belgium. In Dutch.

EVENSEN, G. (2003), The Ensemble Kalman Filter: theoretical formulation and practical implementation. Ocean Dynamics 53(4), 343–367.

EWERT, F., M. K. VAN ITTERSUM, I. BEZLEPKINA, O. THEROND, E. ANDERSEN, H. BELHOUCHETTE, C. BOCKSTALLER, F. BROUWER, T. HECKELEI, S. JANSSEN, R. KNAPEN, M. KUIPER, K. LOUHICHI, J. A. OLSSON, N. TURPIN, J. WERY, J. E. WIEN and J. WOLF (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. and J. FALL (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: M. Bubak, G. van Albada, P. Sloot and J. Dongarra, eds., Computational Science - ICCS 2004, vol. 3039 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1210–1217.

FAO (2003), World agriculture towards 2015/2030 an FAO perspective. Food and Agriculture Organisation .

FENICIA, F., H. H. G. SAVENIJE, P. MATGEN and L. PFISTER (2008), Understanding catchment behavior through stepwise model concept improvement. Water Resources Research 44(1), W01402.

FENICIA, F., D. KAVETSKI and H. H. G. SAVENIJE (2011), Elements of a flexible approach for conceptual hydrological modeling: 1. Motivation and theoretical development. Water Resources Research 47(11), W11510.

FONSECA, F. T., M. J. EGENHOFER, P. AGOURIS and G. CÂMARA (2002), Using Ontologies for Integrated Geographic Information Systems. Transactions in GIS 6(3), 231–257.

FRANK, A. U. (2001), Tiers of ontology and consistency constraints in geographical information systems. International Journal of Geographical Information Science 15(7), 667–678.

FRIEDENTHAL, S., A. MOORE and R. STEINER (2008), A Practical Guide to SysML: Systems Modeling Language. Morgan Kaufmann.

GAMMA, E., R. HELM, R. JOHNSON and J. VLISSIDES (1995), Design patterns: elements of reusable object-oriented software. Addison Wesley.

GANSNER, E. R. and S. C. NORTH (2000), An open graph visualization system and its applications to software engineering. SOFTWARE-PRACTICE AND EXPERIENCE 30(11), 1203–1233.

GDAL DEVELOPMENT TEAM (2013), GDAL - Geospatial Data Abstraction Library. URL http://www.gdal.org/.

GEDEON, M., I. WEMAERE and J. MARIVOET (2007), Regional groundwater model of north-east Belgium. Journal of Hydrology 335(1–2), 133–139.

GLEESON, T., Y. WADA, M. F. P. BIERKENS and L. P. H. VAN BEEK (2012), Water balance of global aquifers revealed by groundwater footprint. Nature 488(7410), 197–200.

GML (2013), Geography Markup Language Encoding Standard. Open Geospatial Consortium. URL http://www.opengeospatial.org/standards/gml.

GOODCHILD, M. F., M. YUAN and T. J. COVA (2007), Towards a general theory of geographic representation in GIS. International Journal of Geographical Information Science 21(3), 239–260.

GRANELL, C., L. DÍAZ, S. SCHADE, N. OSTLÄNDER and J. HUERTA (2013a), Enhancing integrated environmental modelling by designing resource-oriented interfaces. Environmental Modelling & Software 39, 229–246.

GRANELL, C., S. SCHADE and N. OSTLÄNDER (2013b), Seeing the forest through the trees: A review of integrated environmental modelling tools. Computers, Environment and Urban Systems 41, 136–150.

GRASS (2013), Geographic Resources Analysis Support System. URL http://grass.itc.it/.

GREGERSEN, J. B., P. J. A. GIJSBERS and S. J. P. WESTEN (2007), OpenMI: Open modelling interface. Journal of Hydroinformatics 9(3), 175–191.

GRENON, P. and B. SMITH (2004), SNAP and SPAN: Towards Dynamic Spatial Ontology. Spatial Cognition & Computation 4(1), 69–104.

GRIMM, V. and S. F. RAILSBACK (2005), Individual-based modelling and ecology. Princeton: Princeton University Press.

GRIMM, V., U. BERGER, F. BASTIANSEN, S. ELIASSEN, V. GINOT, J. GISKE, J. GOSS-CUSTARD, T. GRAND, S. K. HEINZ, G. HUSE, A. HUTH, J. U. JEPSEN, C. JØRGENSEN, W. M. MOOIJ, B. MÜLLER, G. PE'ER, C. PIOU, S. F. RAILSBACK, A. M. ROBBINS, M. M. ROBBINS, E. ROSSMANITH, N. RÜGER, E. STRAND, S. SOUISSI, R. A. STILLMAN, R. VABØ, U. VISSER and D. L. DEANGELIS (2006), A standard protocol for describing individual-based and agent-based models. Ecological Modelling 198(1–2), 115–126.

GRUNINGER, M. and X. TAN (2009), Reasoning about partially ordered web service activities in PSL. Lecture Notes in Computer Science 5926, 231–245.

GÜNTNER, A., J. OLSSON, A. CALVER and B. GANNON (2001), Cascade-based disaggregation of continuous rainfall time series: The influence of climate. Hydrology and Earth System Sciences 5(2), 145–164.

GURNEY, W. S. C. and R. M. NISBET (1998), Ecological Dynamics. New York: Oxford University Press.

HAASNOOT, M., H. MIDDELKOOP, A. OFFERMANS, E. VAN BEEK and W. P. A. VAN DEURSEN (2012), Exploring pathways for sustainable water management in river deltas in a changing environment. Climatic Change 115(3–4), 795–819.

HAGEN-ZANKER, A., J. VAN LOON, A. MAAS, B. STRAATMAN, T. DE NIJS and G. ENGELEN (2005), Measuring performance of land use models: An evaluation framework for the calibration and validation of integrated land use models featuring cellular automata. In: 14th European Colloquium on Theoretical and Quantitative Geography. Lisbon, Portugal.

HAHN, B. M., S. KOFALK, J.-L. DE KOK, J. BERLEKAMP and M. EVERS (2009), Elbe DSS: A Planning Support System for Strategic River Basin Planning. In: S. Geertman and J. Stillwell, eds., Planning Support Systems Best Practice and New Methods, chap. 6, Springer Netherlands, 113–136.

HALL, J. and D. SOLOMATINE (2008), A framework for uncertainty analysis in flood risk management decisions. International Journal of River Basin Management 6(2), 85–98.

HARBAUGH, A. W., E. R. BANTA, M. C. HILL and M. G. MCDONALD (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.

HARVEY, H., J. HALL and R. PEPPÉ (2012), Computational decision analysis for flood risk management in an uncertain future. Journal of Hydroinformatics 14(3), 537–561.

HDF5 (2010), Hierarchical data format version 5, 2000-2010. The HDF Group. URL http://www.hdfgroup.org/HDF5.

HENNING, M. (2004), A new approach to object-oriented middleware. IEEE Internet Computing 8(1), 66–75.

HERZOG, B. L., D. R. LARSON, C. C. ABERT, S. D. WILSON and G. S. ROADCAP (2003), Hydrostratigraphic modeling of a complex, glacial-drift aquifer system for importation into MODFLOW. Ground Water 41(1), 57–65.

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.

HEUVELINK, G. B. M. (1998), Error Propagation in Environmental Modelling with GIS. London: Taylor & Francis.

HIBBARD, K. A. and A. C. JANETOS (2013), The regional nature of global challenges: a need and strategy for integrated regional modeling. Climatic Change 118(3–4), 565–577.

HIEMSTRA, P. H., D. KARSSENBERG and A. VAN DIJK (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.

HILL, C., C. DELUCA, BALAJIC, M. SUAREZ and A. DA SILVA (2004), The Architecture of the Earth System Modeling Framework. Computing in Science and Engineering 6(1), 18–28.

HILL, M. C. and C. R. TIEDEMAN (2007), Effective Groundwater Model Calibration: With Analysis of Data, Sensitivities, Predictions, and Uncertainty. John Wiley & Sons.

HINKEL, J. (2009), The PIAM approach to modular integrated assessment modelling. Environmental Modelling & Software 24(6), 739–748.

HOLZWORTH, D. P., N. I. HUTH and P. G. DE VOIL (2010), Simplifying environmental model reuse. Environmental Modelling & Software 25(2), 269–275.

HSU, M.-H., A. Y. KUO, J.-T. KUO and W.-C. LIU (1999), Procedure to calibrate and verify numerical models of estuarine hydrodynamics. Journal of Hydraulic Engineering 125(2), 166–182.

HUANG, J., J. GAO, G. HÖRMANN and W. M. MOOIJ (2012), Integrating three lake models into a Phytoplankton Prediction System for Lake Taihu (Taihu PPS) with Python. Journal of Hydroinformatics 14(2), 523–534.

HUANG, M., D. R. MAIDMENT and Y. TIAN (2011), Using SOA and RIAs for water data discovery and retrieval. Environmental Modelling & Software 26(11), 1309–1324.

HUNTER, J. D. (2007), Matplotlib: A 2D graphics environment. Computing in Science and Engineering 9(3), 99–104.

HURKENS, J., B. HAHN and H. VAN DELDEN (2008), Using the GEONAMICA software environment for integrated dynamic spatial modelling. In: M. Sànchez-Marrè, J. Béjar, J. Comas, A. Rizzoli and G. Guariso, eds., Integrating Sciences and Information Technology for Environmental Assessment and Decision Making, vol. 2 of *Proceedings of the iEMSs Fourth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2008)*, International Environmental Modelling and Software Society, Barcelona, Catalonia, 751–758.

ICPR (2001), Abschlussbericht. Vorgehensweise zur Ermittlung der hochwassergefährdeten Flächen und der möglichen Vermögenschäden (Final report. Approach to investigate the areas at flood risk and potential economic damage). Koblenz, Germany.

JAGERS, H. R. A. (2010), Linking data, models and tools: An overview. In: D. A. Swayne, W. Yang, A. A. Voinov, A. Rizzoli and T. Filatova, eds., Modelling for Environment's Sake, vol. 2 of *Proceedings of the iEMSs Fifth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2010)*, International Environmental Modelling and Software Society, Ottawa, Canada, 1150–1157.

JAKEMAN, A. J. and R. A. LETCHER (2003), Integrated assessment and modelling: features, principles and examples for catchment management. Environmental Modelling & Software 18(6), 491–501.

JAKEMAN, A. J., R. A. LETCHER and J. P. NORTON (2006), Ten iterative steps in development and evaluation of environmental models. Environmental Modelling & Software 21(5), 602–614.

JANSSEN, S., F. EWERT, H. LI, I. N. ATHANASIADIS, J. J. F. WIEN, O. THÉROND, M. J. R. KNAPEN, I. BEZLEPKINA, J. ALKAN-OLSSON, A. E. RIZZOLI, H. BELHOUCHETTE, M. SVENSSON and M. K. VAN ITTERSUM (2009), Defining assessment projects and scenarios for policy support: Use of ontology in Integrated Assessment and Modelling. Environmental Modelling & Software 24(12), 1491–1500.

JANSSEN, S., I. N. ATHANASIADIS, I. BEZLEPKINA, R. KNAPEN, H. LI, I. P. DOMÍNGUEZ, A. E. RIZZOLI and M. K. VAN ITTERSUM (2011), Linking models for assessing agricultural land use change. Computers and Electronics in Agriculture 76(2), 148–160.

JOPPICH, W. and M. KÜRSCHNER (2006), MpCCI-a tool for the simulation of coupled applications. Concurrency and Computation: Practice and Experience 18(2), 183–192.

JØRGENSEN, S. E. and G. BENDORICCHIO (2001), Fundamentals of ecological modelling. Amsterdam: Elsevier.

KARAFYLLIDIS, I. and A. THANAILAKIS (1997), A model for predicting forest fire spreading using cellular automata. Ecological Modelling 99(1), 87–97.

KARAOUZAS, I., E. DIMITRIOU, N. SKOULIKIDIS, K. GRITZALIS and E. COLOMBARI (2009), Linking Hydrogeological and Ecological Tools for an Integrated River Catchment Assessment. Environmental Modeling & Assessment 14(6), 677–689.

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

KARSSENBERG, D. (2006), Upscaling of saturated conductivity for Hortonian runoff modelling. Advances in Water Resources 29(5), 735–759.

KARSSENBERG, D. and J. S. BRIDGE (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(6), 1717–1745.

KARSSENBERG, D. and K. DE JONG (2005a), Dynamic environmental modelling in GIS: 1. Modelling in three spatial dimensions. International Journal of Geographical Information Science 19(5), 559–579.

KARSSENBERG, D. and K. DE JONG (2005b), Dynamic environmental modelling in GIS: 2. Modelling error propagation. International Journal of Geographical Information Science 19(6), 623–637.

KARSSENBERG, D., P. A. BURROUGH, R. SLUITER and K. DE JONG (2001), The PCRaster software and course materials for teaching numerical modelling in the environmental sciences. Transactions in GIS 5(2), 99–110.

KARSSENBERG, D., K. DE JONG and J. VAN DER KWAST (2007), Modelling landscape dynamics with Python. International Journal of Geographical Information Science 21(5), 483–495.

KARSSENBERG, D., K. DE JONG and O. SCHMITZ (2008a), A software environment for stochastic spatio-temporal modelling. In: Geophysical Research Abstracts, vol. 10, Vienna, Austria, EGU2008–A–11121.

KARSSENBERG, D., O. SCHMITZ, L. M. DE VRIES and K. DE JONG (2008b), A tool for construction of stochastic spatio-temporal models assimilated with observational data. In: L. Bernard, A. Friss-Christensen, H. Pundt and I. Compte, eds., 11th AGILE International Conference on Geographic Information Science, Girona, Spain.

KARSSENBERG, D., O. SCHMITZ, P. SALAMON, K. DE JONG and M. F. P. BIERKENS (2010), A software framework for construction of process-based stochastic spatio-temporal models and data assimilation. Environmental Modelling & Software 25(4), 489–502.

KAVETSKI, D. and F. FENICIA (2011), Elements of a flexible approach for conceptual hydrological modeling: 2. Application and experimental insights. Water Resources Research 47(11), W11511.

KIEHLE, C., K. GREVE and C. HEIER (2007), Requirements for Next Generation Spatial Data Infrastructures-Standardized Web Based Geoprocessing and Web Service Orchestration. Transactions in GIS 11(6), 819–834.

KILLCOYNE, S. and J. BOYLE (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.

KLEINHANS, M. G., M. F. P. BIERKENS and M. VAN DER PERK (2010), On the use of laboratory experimentation: "Hydrologists, bring out shovels and garden hoses and hit the dirt". Hydrology

and Earth System Sciences 14(2), 369–382.

KNAPEN, R., S. JANSSEN, O. ROOSENSCHOON, P. VERWEIJ, W. DE WINTER, M. UITERWIJK and J.-E. WIEN (2013), Evaluating OpenMI as a model integration platform across disciplines. Environmental Modelling & Software 39, 274–282.

KNIBERG, H. (2011), Lean from the Trenches: Managing Large-Scale Projects with Kanban. Pragmatic Bookshelf.

KOOMSAP, P., N. I. SHAIKH and V. V. PRABHU (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., K. B. VACHÉ, H.-G. FREDE and L. BREUER (2011), CMF: A Hydrological Programming Language Extension For Integrated Catchment Models. Environmental Modelling & Software 26(6), 828–830.

KRAGT, M. E., L. T. H. NEWHAM, J. BENNETT and A. J. JAKEMAN (2011), An integrated approach to linking economic valuation and catchment modelling. Environmental Modelling & Software 26(1), 92–102.

KRAINES, S., R. BATRES, M. KOYAMA, D. WALLACE and H. KOMIYAMA (2005), Internet-Based Integrated Environmental Assessment Using Ontologies to Share Computational Models. Journal of Industrial Ecology 9(3), 31–50.

KRYSANOVA, V., F. HATTERMANN and F. WECHSUNG (2007), Implications of complexity and uncertainty for integrated modelling and impact assessment in river basins. Environmental Modelling & Software 22(5), 701–709.

KUHN, W. (2001), Ontologies in support of activities in geographical space. International Journal of Geographical Information Science 15(7), 613–631.

LANGTANGEN, H. P. (2007), Python Scripting for Computational Science. Springer, 3rd edn.

LANIAK, G. F., J. G. DROPPO, E. R. FAILLACE, E. K. GNANAPRAGASAM, W. B. MILLS, D. L. STRENGE, G. WHELAN and C. YU (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., G. OLCHIN, J. GOODALL, A. VOINOV, M. HILL, P. GLYNN, G. WHELAN, G. GELLER, N. QUINN, M. BLIND, S. PECKHAM, S. REANEY, N. GABER, R. KENNEDY and A. HUGHES (2013), Integrated environmental modeling: A vision and roadmap for the future. Environmental Modelling & Software 39, 3–23.

LARSON, J., R. JACOB and E. ONG (2005), The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models. International Journal of High Performance Computing Applications 19(3), 277–292.

LEAVESLEY, G. H., P. J. RESTREPO, S. L. MARKSTROM, M. DIXON and L. G. STANNARD (1998), The modular modeling system - MMS: User's manual, vol. 96-151. U.S. Geological Survey Open File Report.

LETCHER, R. A., B. F. W. CROKE, W. S. MERRITT and A. J. JAKEMAN (2006), An integrated modelling toolbox for water resources assessment and management in highland catchments: Sensitivity analysis and testing. Agricultural Systems 89(1), 132–164.

LI, A. (2012), Modeling human decisions in coupled human and natural systems: Review of agent-based models. Ecological Modelling 229, 25–36.

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., J. RAUBERG and F. B. HESSER (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.

LINDENSCHMIDT, K.-E., K. FLEISCHBEIN and M. BABOROWSKI (2007), Structural uncertainty in a river water quality modelling system. Ecological Modelling 204(3–4), 289–300.

LIU, J., C. PENG, Q. DANG, M. APPS and H. JIANG (2002), A component object model strategy for reusing ecosystem models. Computers and Electronics in Agriculture 35(1), 17–33.

LIU, J. S. and R. CHEN (1998), Sequential Monte Carlo methods for dynamic systems. Journal of the American Statistical Association 93(443), 1032–1044.

LIU, Y. and H. GUPTA (2007), Uncertainty in hydrologic modeling: Toward an integrated data assimilation framework. Water Resources Research 43(7), W07401.

LIU, Y. B. and F. DE SMEDT (2005), Flood modeling for complex terrain using GIS and remote sensed information. Water Resources Management 19(5), 605–624.

LIU, Y. B., S. GEBREMESKEL, F. D. SMEDT, L. HOFFMANN and L. PFISTER (2003), A diffusive transport approach for flow routing in GIS-based flood modeling. Journal of Hydrology 283(1–4), 91–106.

LLOYD, W., O. DAVID, J. C. ASCOUGH II, K. W. ROJAS, J. R. CARLSON, G. H. LEAVESLEY, P. KRAUSE, T. R. GREEN and L. R. AHUJA (2011), Environmental modeling framework invasiveness: Analysis and implications. Environmental Modelling & Software 26(10), 1240–1250.

LÓPEZ, I. F. V., R. T. SNODGRASS and B. MOON (2005), Spatiotemporal Aggregate Computation: A Survey. IEEE Transactions on Knowledge and Data Engineering 17(2), 271–286.

LOUI, R. P. (2008), In Praise of Scripting: Real Programming Pragmatism. Computer 41, 22–26.

LUTZ, M. and E. KLIEN (2006), Ontology-based retrieval of geographic information. International Journal of Geographical Information Science 20(3), 233–260.

MADIN, J., S. BOWERS, M. SCHILDHAUER, S. KRIVOV, D. PENNINGTON and F. VILLA (2007), An ontology for describing and synthesizing ecological observation data. Ecological Informatics 2(3), 279–296.

MADIN, J. S., S. BOWERS, M. P. SCHILDHAUER and M. B. JONES (2008), Advancing ecological research with ontologies. Trends in Ecology & Evolution 23(3), 159–168.

MAHMOUD, M., Y. LIU, H. HARTMANN, S. STEWART, T. WAGENER, D. SEMMENS, R. STEWART, H. GUPTA, D. DOMINGUEZ, F. DOMINGUEZ, D. HULSE, R. LETCHER, B. RASHLEIGH, C. SMITH, R. STREET, J. TICEHURST, M. TWERY, H. VAN DELDEN, R. WALDICK, D. WHITE and L. WINTER (2009), A formal framework for scenario development in support of environmental decision-making. Environmental Modelling & Software 24(7), 798–808.

MALONE, B. P., A. B. MCBRATNEY, B. MINASNY and I. WHEELER (2012), A general method for downscaling earth resource information. Computers & Geosciences 41, 119–125.

MARSHALL, L., A. SHARMA and D. NOTT (2006), Modeling the catchment via mixtures: Issues of model specification and validation. Water Resources Research 42(11), W11409.

MATLAB (2013), The MathWorks product website. URL http://www.mathworks.com/.

MATOTT, L. S., J. E. BABENDREIER and S. T. PURUCKER (2009), Evaluating uncertainty in integrated environmental models: A review of concepts and tools. Water Resources Research 45(6),

W06421.

MAY, R. M. (1977), Thresholds and breakpoints in ecosystems with a multiplicity of stable states. Nature 269(5628), 471–477.

MCARTHUR, K., H. SAIEDIAN and M. ZAND (2002), An evaluation of the impact of component-based architectures on software reusability. Information and Software Technology 44(6), 351–359.

MCDONALD, M. G. and A. W. HARBAUGH (1984), A modular three-dimensional user's guide for finite-difference ground-water flow model. USGS.

MCINTOSH, B. S., P. JEFFREY, M. LEMON and N. WINDER (2005), On the design of computer-based models for integrated environmental science. Environmental Management 35(6), 741–752.

MCINTOSH, B. S., R. A. F. SEATON and P. JEFFREY (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.

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. and M. AIVAZIS (2011), Python for scientists and engineers. Computing in Science and Engineering 13(2), 9–12.

MINETER, M. J., C. H. JARVIS and S. DOWERS (2003), From stand-alone programs towards grid–aware services and components: a case study in agricultural modelling with interpolated climate data. Environmental Modelling & Software 18(4), 379–391.

MIRALLES, A., F. PINET and Y. BÉDARD (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.

MONNINKHOFF, B. and K. KERNBACH (2006), Coupled surface water - groundwater modelling for planning of flood retention in the lower havel area. In: Proceedings of the International FEFLOW User Conference, 115–124.

MOORE, R. V. and C. I. TINDALL (2005), An overview of the open modelling interface and environment (the OpenMI). Environmental Science & Policy 8(3), 279–286.

MORADKHANI, H., K.-L. HSU, H. GUPTA and S. SOROOSHIAN (2005), Uncertainty assessment of hydrologic model states and parameters: Sequential data assimilation using the particle filter. Water Resources Research 41(5). W05012.

MOREIRA, E., S. COSTA, A. AGUIAR, G. CÂMARA and T. CARNEIRO (2009), Dynamical coupling of multiscale land change models. Landscape Ecology 24(9), 1183–1194.

MORIASI, D. N., J. G. ARNOLD, M. W. VAN LIEW, R. L. BINGNER, R. D. HARMEL and T. L. VEITH (2007), Model evaluation guidelines for systematic quantification of accuracy in watershed simulations. Transactions of the ASABE 50(3), 885–900.

MORITA, M. and B. C. YEN (2002), Modeling of Conjunctive Two-Dimensional Surface-Three-Dimensional Subsurface Flows. Journal of Hydraulic Engineering 128(2), 184–200.

MUETZELFELDT, R. and J. MASSHEDER (2003), The Simile visual modelling environment. European Journal of Agronomy 18(3–4), 345–358.

MUTH JR., D. J. and K. M. BRYDEN (2013), An integrated model for assessment of sustainable agricultural residue removal limits for bioenergy systems. Environmental Modelling & Software 39, 50–69.

NETCDF-CF (2013), NetCDF Climate and Forecast (CF) Metadata Convention. URL http://cf-pcmdi.llnl.gov/.

NGUYEN, C. D., H. ARAKI, H. YAMANISHI and K. KOGA (2005), Simulation of groundwater flow and environmental effects resulting from pumping. Environmental Geology 47(3), 361–374.

NOGUERAS-ISO, J., F. J. ZARAZAGA-SORIA, J. LACASTA, R. BÉJAR and P. R. MURO-MEDRANO (2004), Metadata standard interoperability: application in the geographic information domain. Computers, Environment and Urban Systems 28(6), 611–634.

NORTH, M. J., N. T. COLLIER and J. R. VOS (2006), Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Transactions on Modeling and Computer Simulation 16(1), 1–25.

OC GIS-VLAANDEREN (2001), Digital version of Land-cover data set of Flanders 2001. Agentschap voor Geografische Informatie Vlaanderen, Ghent, Belgium.

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.

OPENDA (2013), The OpenDA data assimilation toolbox website. URL http://www.openda.org/.

OPENMI (2013), Open Modelling Interface website. URL http://www.openmi.org/.

OPENMI 2 SDK (2013), Open Modelling Interface Software Development Kit. URL http://www.openmi.org/.

ORESKES, N., K. SHRADER-FRECHETTE and K. BELITZ (1994), Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. Science 263(5147), 641–646.

OSMAN, Y. Z. and M. P. BRUEN (2002), Modelling stream - Aquifer seepage in an alluvial aquifer: An improved loosing-stream package for MODFLOW. Journal of Hydrology 264(1–4), 69–86.

OWL (2004), OWL Web Ontology Language Guide. URL http://www.w3.org/TR/owl-guide/.

OXLEY, T., P. JEFFREY and M. LEMON (2002), Policy relevant modelling: relationships between water, land use and farmer decision processes. Integrated Assessment 3(1), 30–49.

OXLEY, T., B. S. MCINTOSH, N. WINDER, M. MULLIGAN and G. ENGELEN (2004), Integrated modelling and decision-support tools: a Mediterranean example. Environmental Modelling & Software 19(11), 999–1010.

PAPAJORGJI, P. (2005), A plug and play approach for developing environmental models. Environmental Modelling & Software 20(10), 1353–1357.

PARNAS, D. L. (1972), On the Criteria to Be Used in Decomposing Systems into Modules. Communications of the ACM 15(12), 1053–1058.

PARTIDÁRIO, M. R. (2000), Elements of an SEA framework–improving the added-value of SEA. Environmental Impact Assessment Review 20(6), 647–663.

PCRASTER (2013), PCRaster website. URL http://www.pcraster.eu.

PEBESMA, E. J., K. DE JONG and D. BRIGGS (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.

PECKHAM, S. D., E. W. H. HUTTON and B. NORRIS (2013), A component-based approach to integrated modeling in the geosciences: The design of CSDMS. Computers & Geosciences 53, 3–12.

PÉREZ, F. and B. E. GRANGER (2007), IPython: A system for interactive scientific computing. Computing in Science and Engineering 9(3), 21–29.

PÉREZ, F., B. E. GRANGER and J. D. HUNTER (2011), Python: An ecosystem for scientific computing. Computing in Science and Engineering 13(2), 13–21.

PEREZ, L. and S. DRAGICEVIC (2012), Landscape-level simulation of forest insect disturbance: Coupling swarm intelligent agents with GIS-based cellular automata model. Ecological Modelling 231, 53–64.

PETERSON, P. (2009), F2PY: A tool for connecting Fortran and Python programs. International Journal of Computational Science and Engineering 4(4), 296–305.

POELMANS, L., A. VAN ROMPAEY and O. BATELAAN (2010), Coupling urban expansion models and hydrological models: How important are spatial patterns? Land Use Policy 27(3), 965–975.

POLHILL, J. G. and N. M. GOTTS (2009), Ontologies for transparent integrated human-natural system modelling. Landscape Ecology 24(9), 1255–1267.

PRECHELT, L. (2000), Empirical comparison of seven programming languages. Computer 33(10), 23–29.

PROCESSING MODFLOW (2013), Processing Modflow website. URL http://www.pmwin.net/.

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.

PULLAR, D. (2004), SimuMap: a computational system for spatial modelling. Environmental Modelling & Software 19(3), 235–243.

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.

R DEVELOPMENT CORE TEAM (2013), R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

RAHMAN, J. M., S. M. CUDDY and F. G. R. WATSON (2004), Tarsier and ICMS: two approaches to framework development. Mathematics and Computers in Simulation 64(3-4), 339–350.

RASINMÄKI, J. (2009), XQuery as a retrieval mechanism for longitudinal multiscale forest resource data. Environmental Modelling & Software 24(10), 1153–1162.

RASINMÄKI, J., A. MÄKINEN and J. KALLIOVIRTA (2009), SIMO: An adaptable simulation framework for multiscale forest resource data. Computers and Electronics in Agriculture 66(1), 76–84.

RASTETTER, E. B., J. D. ABER, D. P. C. PETERS, D. S. OJIMA and I. C. BURKE (2003), Using Mechanistic Models to Scale Ecological Processes across Space and Time. BioScience 53(1), 68–76.

RDF (2004), Resource Description Framework (RDF). URL http://www.w3.org/RDF/.

REFSGAARD, J. C. and H. J. HENRIKSEN (2004), Modelling guidelines - terminology and guiding principles. Advances in Water Resources 27(1), 71–82.

REFSGAARD, J. C., J. P. VAN DER SLUIJS, J. BROWN and P. VAN DER KEUR (2006), A framework for dealing with uncertainty due to model structure error. Advances in Water Resources 29(11), 1586–1597.

REFSGAARD, J. C., J. P. VAN DER SLUIJS, A. L. HØJBERG and P. A. VANROLLEGHEM (2007), Uncertainty in the environmental modelling process - A framework and guidance. Environmental Modelling & Software 22(11), 1543–1556.

REPAST (2013), Recursive Porous Agent Simulation Toolkit website. URL http://repast.sourceforge.net/.

RISBEY, J., J. SLUIJS, P. KLOPROGGE, J. RAVETZ, S. FUNTOWICZ and S. CORRAL QUINTANA (2005), Application of a checklist for quality assistance in environmental modelling to an energy model. Environmental Modeling & Assessment 10(1), 63–79.

RIZZOLI, A. E. and R. M. ARGENT (2006), Software and Software Systems: Platforms and Issues for IWRM Problems. In: C. Giupponi, D. Karssenberg, A. Jakeman and M. P. Hare, eds., Sustainable Management of Water Resources: An Integrated Approach, chap. 12, Edward Elgar Publishing, 324–346.

RIZZOLI, A. E., M. DONATELLI, I. N. ATHANASIADIS, F. VILLA and D. HUBER (2008), Semantic links in integrated modelling frameworks. Mathematics and Computers in Simulation 78(2–3), 412–423.

ROBERTS, J. J., B. D. BEST, D. C. DUNN, E. A. TREML and P. N. HALPIN (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.

ROTMANS, J. (1990), IMAGE: an integrated model to assess the greenhouse effect. Kluwer.

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. Boston, MA: Addison-Wesley.

RYKIEL, E. J. (1996), Testing ecological models: the meaning of validation. Ecological Modelling 90(3), 229–244.

SAFARI, A., F. DE SMEDT and F. MOREDA (2012), WetSpa model application in the Distributed Model Intercomparison Project (DMIP2). Journal of Hydrology 418–419, 78–89.

SAIFUL ISLAM, A. K. M. and M. PIASECKI (2006), A generic metadata description for hydrodynamic model data. Journal of Hydroinformatics 8(2), 141–148.

SARGENT, R. G. (2013), Verification and validation of simulation models. Journal of Simulation 7(1), 12–24.

SCHELLER, R. M., B. R. STURTEVANT, E. J. GUSTAFSON, B. C. WARD and D. J. MLADENOFF (2010), Increasing the reliability of ecological models using modern software engineering techniques. Frontiers in Ecology and the Environment 8(5), 253–260.

SCHMITZ, O., D. KARSSENBERG and J.-L. DE KOK (2012), Towards integrated model building with semantically annotated components. In: R. Seppelt, A. A. Voinov, S. Lange and D. Bankamp, eds., Managing Resources of a Limited Planet: Pathways and Visions under Uncertainty, Proceedings of the iEMSs Sixth Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2012), International Environmental Modelling and Software Society, Leipzig, Germany, 2403–2412.

SCHMOLKE, A., P. THORBEK, D. L. DEANGELIS and V. GRIMM (2010), Ecological models supporting environmental decision making: a strategy for the future. Trends in Ecology & Evolution 25(8), 479–486.

SCHOLTEN, H., A. KASSAHUN, J. C. REFSGAARD, T. KARGAS, C. GAVARDINAS and A. J. BEULENS (2007), A methodology to support multidisciplinary model-based water management. Environ-

mental Modelling & Software 22(5), 743–759.

SCHROEDER, W. J., L. S. AVILA and W. HOFFMAN (2000), Visualizing with VTK: A tutorial. IEEE Computer Graphics and Applications 20(5), 20–27.

SCHWANGHART, W. and N. J. KUHN (2010), TopoToolbox: A set of Matlab functions for topographic analysis. Environmental Modelling & Software 25(6), 770–781.

SCHWEITZER, C., J. A. PRIESS and S. DAS (2011), A generic framework for land-use modelling. Environmental Modelling & Software 26(8), 1052–1055.

SCIPY (2013), Scientific Tools for Python. URL http://www.scipy.org/.

SIMON, D. (2006), Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches. John Wiley & Sons.

SKLAR, E. (2007), Software review: NetLogo, a multi-agent simulation environment. Artificial Life 13(3), 303–311.

SKØIEN, J. O., G. BLÖSCHL and A. W. WESTERN (2003), Characteristic space scales and timescales in hydrology. Water Resources Research 39(10), SWC111–SWC1119.

SPHINX (2013), Python documentation generator. URL http://sphinx-doc.org/.

STASCH, C., T. FOERSTER, C. AUTERMANN and E. PEBESMA (2012), Spatio-temporal aggregation of European air quality observations in the Sensor Web. Computers & Geosciences 47, 111–118.

STEINIGER, S. and G. J. HAY (2009), Free and open source geographic information tools for landscape ecology. Ecological Informatics 4(4), 183–195.

STELLA (2013), STELLA product website. URL http://www.iseesystems.com/.

SUTANUDJAJA, E. H. (2012), The use of soil moisture - remote sensing products for large-scale groundwater modeling and assessment, vol. 25. Utrecht Studies in Earth Sciences.

SUTANUDJAJA, E. H., L. P. H. VAN BEEK, S. M. DE JONG, F. C. VAN GEER and M. F. P. BIERKENS (2014), Calibrating a large-extent high-resolution coupled groundwater-land surface model using soil moisture and discharge data. Water Resources Research 50(1), 687–705.

SZYPERSKI, C. (2002), Component Software: Beyond Object-Oriented Programming. Addison Wesley, 2nd edn.

TANENBAUM, A. S. (1992), Modern operating systems. Prentice-Hall.

TDT (2010), Typed Data Transfer (TDT) Library website. URL http://www.pik-potsdam.de/software/tdt.

TOMLIN, C. D. (1990), Geographic Information Systems and Cartographic Modeling. Englewood Cliffs, NJ: Prentice Hall.

TORREY, L. A., J. COLEMAN and B. P. MILLER (2007), A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. Software-Practice and Experience 37(4), 347–364.

TUCKER, G. E., S. T. LANCASTER, N. M. GASPARINI, R. L. BRAS and S. M. RYBARCZYK (2001), An object-oriented framework for distributed hydrologic and geomorphic modeling using triangulated irregular networks. Computers & Geosciences 27(8), 959–973.

UNITED NATIONS (2002), Report of the World Summit on Sustainable Development. United Nations, New York .

USCHOLD, M. and M. GRUNINGER (1996), Ontologies: Principles, methods and applications. Knowledge Engineering Review 11(2), 93–136.

USGS (2013), US Geologic Survey. URL http://water.usgs.gov/nrp/gwsoftware/modflow.html.

VAN BEEK, L. P. H., Y. WADA and M. F. P. BIERKENS (2011), Global monthly water stress: 1. Water balance and water availability. Water Resources Research 47(7), W07517.

VAN BEURDEN, A. U. C. J. and W. J. A. M. DOUVEN (1999), Aggregation issues of spatial information in environmental research. International Journal of Geographical Information Science 13(5), 513–527.

VAN DELDEN, H., P. LUJA and G. ENGELEN (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., J. YOUNIS and A. P. J. DE ROO (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., S. C. COLBERT and G. VAROQUAUX (2011), The NumPy array: A structure for efficient numerical computation. Computing in Science and Engineering 13(2), 22–30.

VAN DEURSEN, W. and J. MAES (2008), Progress on PCRaster-Extend Interface (Update Report), Deliverable 8.6, Workpackage 8, Model Support. Tech. Rep., Science Policy Interface for Coastal Systems Assessment (SPICOSA), EU 6th Framework Research Project nr. 0369927.

VAN DEURSEN, W. P. A. (1995), Geographical Information Systems and Dynamic Models. Utrecht: Koninklijk Nederlands Aardrijkskundig Genootschap/Faculteit Ruimtelijke Wetenschappen, Universiteit Utrecht.

VAN ITTERSUM, M. K., F. EWERT, T. HECKELEI, J. WERY, J. ALKAN OLSSON, E. ANDERSEN, I. BEZLEPKINA, F. BROUWER, M. DONATELLI, G. FLICHMAN, L. OLSSON, A. E. RIZZOLI, T. VAN DER WAL, J. E. WIEN and J. WOLF (2008), Integrated assessment of agricultural systems - A component-based framework for the European Union (SEAMLESS). Agricultural Systems 96(1–3), 150–165.

VAN LEEUWEN, P. J. (2003), A Variance-Minimizing Filter for Large-Scale Applications. Monthly Weather Review 131(9), 2071–2084.

VAN OIJEN, M. (2002), On the use of specific publication criteria for papers on process-based modelling in plant science. Field Crops Research 74(2–3), 197–205.

VERBURG, P. (2006), Simulating feedbacks in land use and land cover change models. Landscape Ecology 21(8), 1171–1183.

VERMAAT, J. E., F. EPPINK, J. C. J. M. VAN DEN BERGH, A. BARENDREGT and J. VAN BELLE (2005), Aggregation and the matching of scales in spatial economics and landscape ecology: empirical evidence and prospects for integration. Ecological Economics 52(2), 229–237.

VERSTEGEN, J. A., D. KARSSENBERG, F. VAN DER HILST and A. FAAIJ (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., M. J. R. KNAPEN, W. P. DE WINTER, J. J. F. WIEN, J. A. TE ROLLER, S. SIEBER and J. M. L. JANSEN (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. and R. COSTANZA (2000), Design of multi-paradigm integrating modelling tools for ecological research. Environmental Modelling & Software 15(2), 169–177.

VILLA, F., I. N. ATHANASIADIS and A. E. RIZZOLI (2009), Modelling with knowledge: A review of emerging semantic approaches to environmental modelling. Environmental Modelling & Software 24(5), 577–587.

VISSER, S. M., G. STERK and D. KARSSENBERG (2004), Wind erosion modelling in a Sahelian environment. Environmental Modelling & Software 20(1), 69–84.

VISUALMODFLOW (2013), Visual Modflow product website. URL http://www.visualmodflow.com/.

VOINOV, A. and C. CERCO (2010), Model integration and the role of data. Environmental Modelling & Software 25(8), 965–969.

VOINOV, A., C. FITZ, R. BOUMANS and R. COSTANZA (2004), Modular ecosystem modeling. Environmental Modelling & Software 19(3), 285–304.

WAINWRIGHT, J. and M. MULLIGAN (2004), Environmental Modelling. Chichester: Wiley.

WANG, X. and D. PULLAR (2005), Describing dynamic modeling for landscapes with vector map algebra in GIS. Computers & Geosciences 31(8), 956–967.

WANG, Z.-M., O. BATELAAN and F. DE SMEDT (1996), A distributed model for water and energy transfer between soil, plants and atmosphere (WetSpa). Physics and Chemistry of the Earth 21(3 SPEC. ISS.), 189–193.

WARD, P. J., H. DE MOEL and J. C. J. H. AERTS (2011), How are flood risk estimates affected by the choice of return-periods? Natural Hazards and Earth System Science 11(12), 3181–3195.

WARNER, J. C., N. PERLIN and E. D. SKYLLINGSTAD (2008), Using the Model Coupling Toolkit to couple earth system models. Environmental Modelling & Software 23(10–11), 1240–1249.

WATERML (2013), WaterML 2.0 Standards Working Group. URL http://www.opengeospatial.org/projects/groups/waterml2.0swg.

WATSON, F. G. R. and J. M. RAHMAN (2004), Tarsier: a practical software framework for model development, testing and deployment. Environmental Modelling & Software 19(3), 245–260.

WEERTS, A. H. and G. Y. H. EL SERAFY (2006), Particle filtering and ensemble Kalman filtering for state updating with hydrological conceptual rainfall-runoff models. Water Resources Research 42, W09403.

WELSH, W. D., J. VAZE, D. DUTTA, D. RASSAM, J. M. RAHMAN, I. D. JOLLY, P. WALLBRINK, G. M. PODGER, M. BETHUNE, M. J. HARDY, J. TENG and J. LERAT (2013), An integrated modelling framework for regulated river systems. Environmental Modelling & Software 39, 81–102.

WERNER, M., J. SCHELLEKENS, P. GIJSBERS, M. VAN DIJK, O. VAN DEN AKKER and K. HEYNERT (2013), The Delft-FEWS flow forecasting system. Environmental Modelling & Software 40, 65–77.

WESSELING, C. G., D. KARSSENBERG, P. A. BURROUGH and W. P. A. VAN DEURSEN (1996), Integrating dynamic environmental models in GIS: The development of a Dynamic Modelling language. Transactions in GIS 1(1), 40–48.

WHITE, R., G. ENGELEN and I. ULJEE (1997), The use of constrained cellular automata for high-resolution modelling of urban land-use dynamics. Environment and Planning B: Planning and Design 24(3), 323–343.

WHITE, R., I. ULJEE and G. ENGELEN (2012), Integrated modelling of population, employment and land-use change with a multiple activity-based variable grid cellular automaton. International Journal of Geographical Information Science 26(7), 1251–1280.

WILLIAMS, M., D. CORNFORD, L. BASTIN and E. J. PEBESMA (2009), Uncertainty Markup Language (UncertML). OGC Discussion Paper, Document Number: 08-122r1.

WINSTON, R. B. (1999), MODFLOW-related freeware and shareware resources on the internet. Computers & Geosciences 25(4), 377–382.

XML (2008), Extensible Markup Language (XML) 1.0. W3C. URL http://www.w3.org/TR/REC-xml/.

ZEIGLER, B. P., H. PRAEHOFER and T. G. KIM (2000), Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press, 2nd edn.

ZHAO, G., G. HÖRMANN, N. FOHRER, J. GAO, H. LI and P. TIAN (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.

Acknowledgements

If you are successful, continue. If you are not successful, continue[1]. Many of my colleagues have helped me to continue and to advance to this point. Foremost I thank my supervisors Steven de Jong and Marc Bierkens for their advice and support throughout the years. I acknowledge my daily supervisors Derek Karssenberg, Jean–Luc de Kok, Kor de Jong and Johannes van der Kwast for their patience, their enthusiasm and many inspiring discussions.

I thank VITO for the opportunity to execute this joint PhD research project. By spending half of the time in Mol, Belgium, I gained interesting insights into their applied research projects, and broadened even further my international experience. The Environmental Modelling Unit provided a kind second working environment. I especially thank Guy Engelen, Inge Uljee, Lien Poelmans, Leen Van Esch, Tomas Crols, Maarten van der Meulen, Klaartje Verbeeck, Elga Salvadore, Jan Bronders, Steven Broekx, Irina Nikolova, Allistair Beames, Praveen Pandey, Roger White, Alexander Herzig, Fraser Morgan, and Daniel Rutledge.

The research groups at the Department of Physical Geography in Utrecht provide an instructive and pleasant working environment. Special thanks to Jon Olav Skøien, Liesbeth Wilschut and Koko Alberti for being great roommates. I also thank my colleagues Noemí Lana–Renault, Lucie Babel, Wiebe Nijland, Paul Hiemstra, Arien Lam, Dominik Wisser, Juul Beltman, Kim Cohen, Maarten Zeylmans van Emmichoven, Niels Drost, Merijn de Bakker, Ekkamol Vannametee, Inge de Graaf, Rens van Beek, Yoshihide Wada, Sebastian Huizer, Maria Stergiadi, Aris Lourens, Menno Straatsma, Niko Wanders, Edwin Sutanudjaja, and Judith Verstegen. I also thank Cees Wesseling and Willem van Deursen from the PCRaster team and Georg Hörmann from Kiel University.

The *BonteCompetitie* in Utrecht is acknowledged for providing great weekends and integrating me into the Dutch football life. Thanks to Frans Opveld, Hans Oude Ophuis, Jos ten Barge, Lex Bruining, Robert Jan Schumacher, Ernst Wark, Marco Goes, Wesley Huisman, Joost Rossdorf, Sem Borrel, Stan Opveld, Rob van der Werff, Klaas Jan Lantinga, Maurice Stakenborg, Henk Pak, Harry Boerma, and Robert Folger.

Finally, thanks to my family for their enduring support.

Oliver Schmitz
Utrecht, March 2014

[1] Source quotation unknown.

Curriculum Vitae

Oliver Schmitz was born on March 27, 1975 in Husum, Germany. He studied computer science at the Christian–Albrechts–Universität in Kiel and obtained his *Diplom* (equivalent to MSc) in 2003. He gained first experience with the more applied disciplines of environmental modelling and geocomputation during the 'Environmental Management' study, a master program that he started in Kiel in 2004. In 2006, he moved to the Netherlands to write his master thesis and to start as a junior researcher at the Department of Physical Geography at Utrecht University. He obtained his second MSc from Kiel University in 2007, and continued to work in Utrecht as a junior researcher on software tools for model coupling, stochastic modelling, and data assimilation. Integrated modelling became the topic of his PhD research, which he started in autumn 2009 as a joint project between Utrecht University and the Environmental Modelling Unit of the Flemish Institute for Technological Research (VITO) in Mol, Belgium. Since October 2013, he has been working as a scientific software engineer at the Department of Physical Geography, Utrecht University.

List of publications

Peer reviewed articles

SCHMITZ, O., DE KOK, J.-L., DE JONG, K. AND KARSSENBERG, D., Formalising component interfaces for building integrated models. In preparation.

SCHMITZ, O., DE KOK, J.-L., DE JONG, K. AND KARSSENBERG, D., Request–reply execution of coupled multi–scale component models. In preparation.

SCHMITZ, O., SALVADORE, E., POELMANS, L., VAN DER KWAST, J. AND KARSSENBERG, D. (in press), A framework to resolve spatio–temporal misalignment in component–based modelling, Journal of Hydroinformatics.

SCHMITZ, O., KARSSENBERG, D., DE JONG, K., DE KOK, J.-L. AND DE JONG, S. M. (2013), Map algebra and model algebra for integrated model building, Environmental Modelling & Software 48, 113–128.

KARSSENBERG, D., SCHMITZ, O., SALAMON, P., DE JONG, K. AND BIERKENS, M. F. P. (2010), A software framework for construction of process–based stochastic spatio–temporal models assimilated with observational data, Environmental Modelling & Software 25(4), 489–502.

SCHMITZ, O., KARSSENBERG, D., VAN DEURSEN, W. P. A. AND WESSELING, C. G. (2009), Linking external components to a spatio–temporal modelling framework: coupling MODFLOW and PCRaster, Environmental Modelling & Software 24(9), 1088–1099.

Conference proceedings

BATELAAN, O., SALVADORE, E., BRONDERS, J. AND SCHMITZ, O. (2013), Urban development and the water balance: coupling land use dynamics and the hydrological system. 20th International Congress on Modelling and Simulation (MODSIM), Adelaide, Australia.

SCHMITZ, O., KARSSENBERG, D. AND DE KOK, J.-L. (2012), Towards integrated model building with semantically annotated components. International Congress on Environmental Modelling and Software. Managing Resources of a Limited Planet: Pathways and Visions under Uncertainty, 6th Biennial Meeting, Leipzig, Germany. International Environmental Modelling and Software Society (iEMSs).

SCHMITZ, O., KARSSENBERG, D., DE JONG, K. AND DE KOK, J.-L. (2011), Constructing integrated models: a scheduler to execute coupled components. 14th AGILE International Conference on Geographic Information Science.

SALAMON, P., KARSSENBERG, D. AND SCHMITZ, O. (2009), Assimilating remotely sensed and other measured data types to improve predictive capabilities of distributed hydrological models using particle filtering. In Sustaining the Millennium Development Goals, 33rd International Symposium on Remote Sensing of Environment, Stresa, Italy.

KARSSENBERG, D., SCHMITZ, O., DE VRIES, L. M. AND DE JONG, K. (2008), A tool for construction of stochastic spatio–temporal models assimilated with observational data. 11th AGILE International Conference on Geographic Information Science, Girona, Spain.

Conference abstracts

SALVADORE, E., BRONDERS, J., SCHMITZ, O. AND BATELAAN, O. (2013), Enhanced flexibility and coupling opportunities: the process–based, open–source, hydrological–model WetSpa. 2nd OpenWater symposium and workshops, Brussels, Belgium.

KARSSENBERG, D., DROST, N., SCHMITZ, O., DE JONG, K. AND BIERKENS, M. F. P. (2013), Large scale stochastic spatio–temporal modelling with PCRaster. Geophysical Research Abstracts, Volume 15, EGU2013-10337, Vienna, Austria.

KARSSENBERG, D., SCHMITZ, O. AND DE JONG, K. (2012), Stochastic spatio–temporal modelling with PCRaster Python. Geophysical Research Abstracts, Volume 14, EGU2012-4367, Vienna, Austria.

SCHMITZ, O., KARSSENBERG, D., DE JONG, K. AND DE KOK, J.-L. (2011), Constructing integrated models: a scheduler to execute coupled components. Geophysical Research Abstracts, Volume 13, EGU2011-9753, Vienna, Austria.

KARSSENBERG, D., DE JONG, K. AND SCHMITZ, O. (2008), A software environment for stochastic spatio-temporal modelling. Geophysical Research Abstracts, Volume 10, EGU2008-A-11121, Vienna, Austria.

SCHMITZ, O., KARSSENBERG, D., VAN DEURSEN, W. P. A. AND BOGAARD, T. (2007), Integrated, exploratory catchment modelling: coupling PCRaster and MODFLOW. Geophysical Research Abstracts, Volume 9, EGU2007-A-09818, Vienna, Austria.