Constructing integrated models: a scheduler to execute coupled components

Oliver Schmitz[1,2], Derek Karssenberg[1], Kor de Jong[1], and Jean-Luc de Kok[2]

[1]Utrecht University, Department of Physical Geography, The Netherlands
[2]Flemish Institute for Technological Research (VITO), Belgium

ABSTRACT

A seamless construction, coupling and modification of model components is important for the development and assessment of integrated models. However, present software frameworks are either tailored to component construction or to component coupling, whereas a consolidated environment is desired. The frameworks also require profound knowledge in system programming languages, and offer limited generic support for model assessment.

We present a software framework for a straightforward construction, coupling, execution and analysis of model components. In this paper, we focus on the treatment of temporal dependencies between components of fixed and variable time step lengths as well as components with confined lifetime. By utilising the high-level scripting language Python, non-software developers are able to conduct exploratory model construction and analysis.

INTRODUCTION

Assembling multiple interacting model components of various domains such as environmental, social and economic systems is known as integrated modelling (Argent, 2004: Hinkel, 2009). The purpose of these integrated models is to obtain a holistic understanding of the behaviour of complex systems not only because of scientific interest (e.g., Liu et al., 2002: Rotmans, 1990: Villa and Costanza, 2000: Voinov et al., 2004), but also driven by policy and management (e.g., Jakeman and Letcher, 2003: Letcher et al., 2007: Parker et al., 2002: Rivington et al., 2007: Engelen, 2004: de Kok et al., 2010). Given that the incorporation of model components from several disciplines and their interaction increases the complexity of models, the use of modular components to assemble integrated models increases the maintainability and can streamline the model construction process. Proper modularisation is reached by the construction of components with well defined input and output interfaces and hidden process implementation.

The concept of modularisation and reuse is ubiquitous in the software engineering domain (e.g., Szyperski, 2002: Booch et al., 2007: Gamma et al., 1995) and influenced the development of software frameworks that support the development of environmental models (Rizzoli et al., 2008: Bian, 2007: Donatelli and Rizzoli, 2008). Existing software frameworks are for example ESMF (Collins et al., 2005), MapScript (Pullar, 2003), Tarsier (Watson and Rahman, 2004), PCRaster (Wesseling et al., 1996) or E2 (Argent et al., 2009). Each of these frameworks provide library functions for easier construction, execution and visualisation of field-based models. Repast (North et al., 2006) or NetLogo (Sklar, 2007) are frameworks providing such a functionality for agent-based models. Contrary to the construction of model components, the Typed Data Transfer (Hinkel, 2009) provides library functions for the exchange of data between components; and the Open Modelling Interface (OpenMI, Gregersen et al., 2007) provides a standardised interface to describe and transfer data between existing components.

Coupled models consist of various components, often with different spatial and temporal properties. The spatial discretisation can differ between components, as for example the coupling of a catchment component with ha resolution to a catchment component with km resolution. Also, components can hold different temporal resolutions, such as fixed time step lengths of one day for a component providing recharge data. In addition, modelling situations can occur with a varying time step, such as in a component describing the seasonal growth of crop with a short time step in the growth period, and a larger one in the fallow time. Moreover, components with undetermined start and end times are common use cases. Examples are an avalanche component triggered at a specific snow load, or components modelling individuals such as moving animals. A coupled model therefore will result in a complex meshwork of spatial and temporal dependencies, including intermediate adapter steps for spatial data conversion and temporal aggregation. However, applying the different existing software frameworks for the construction and coupling of components is not straightforward.

The aim of this study is to develop a software framework for the construction of components, their coupling as well as the assessment of components and coupled models as a whole. Therefore, we define the following questions:

- How can the temporal dependencies be derived between model components that exchange data, and how can these be represented by an ordered execution scheme?
- How can the model developer construct and schedule components without making use of system programming languages?

First, we define the requirements for model components and scheduling schemes with the help of common modelling scenarios. From these requirements we derive the execution schemes that need to be implemented within the scheduler. After the description of the software prototype, we outline remaining issues and further directions in this research.

MODELLING SITUATIONS AND IMPLICATIONS FOR THE SCHEDULING OF COUPLED COMPONENTS

In this section, we derive the requirements for the scheduling schemes and the model components based on examples of coupling scenarios with different temporal discretisations. First, we outline the scenario of coupling components with fixed time steps. After that, components with variable time steps are described. Finally, the incorporation of components with an undetermined start and end time is described.

Fixed time steps for all components

First, we consider a situation in which all components in the model have a time step that is known at the model development stage and fixed during the model execution. In addition to different time steps, several component interactions can occur in a coupled model. Components may not interact at all, interact in one direction, or interact bidirectional. Each situation brings along different requirements in terms of the data exchange between the components.

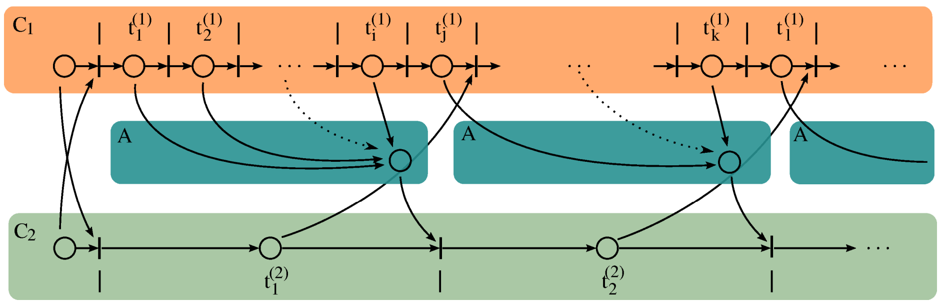

Figure 1: Temporal control flow between interacting components with shorter (C_1) and longer (C_2) time steps. The circles represent the state of the component at each time step, for example the amount of percolation or seepage. The bars represent the transition from time step t_n to $_{tn+1}$, that is the calculation of the processes described within that component. The component request the most recent data available. While C_1 directly accepts the input of C_2, C_2 expects aggregated values such as monthly mean values from C_1, provided by adapter A. Note that data conversion issues is not explicitly included in the graph.

Figure 1 shows the case of a bidirectional coupling between a component C_1 calculating a process such as soil water percolation and a component C_2 calculating for example groundwater flow. Here, the component C_1 calculates with a fixed daily time step whereas C_2 calculates with a fixed time step of one month. As a consequence, 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 is not representative for use in C_2. Therefore, an aggregated value needs to be calculated before it can be passed to C_2, which is done by an adapter. While most modelling situations also require an adapter in the opposite direction, we here simplify by C_1 directly accepting the latest seepage values from C_2.

In general, the order of component execution is arbitrary as long as the dependencies are met at the data exchange moments. The scheduling could be as follows. From its initial state, C_2 obtains data from C_1 and propagates to its new state in $t_1^{(2)}$. As C_2 expects new input data to proceed to its next time step that is not yet available, C_2 exports its current state and waits until the adapter A can provide the aggregated values of 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 $t_1^{(1)}$. As the data exchange moment with C_2 is not yet reached, C_1 can proceed further by reusing the initial value of C_2 until the data exchange moment. In addition, the states of each time step need to be stored in order to process them in the adapter. C_1 can continue after the data exchange moment because C_2 already can provide recent data of its latest time step. C_2 can proceed to its next state in $t_2^{(2)}$ as soon as all values from C_1 are available for and processed by the adapter.

To determine a proper order of component execution and data exchange moments the scheduler needs to obtain information from all coupled components in the model. All process components need to be registered with the start and end time as well as the time step duration. This information is used to build up a shared timeline of all components. Furthermore, the interaction between components needs to be expressed, i.e. which components exchange data. From this specification, the synchronisation moments between the components can be derived. In case of aggregated values over time or data conversion, adapter need to be declared. The scheduler also determines components that do not exchange data until a synchronization moment. These components can proceed independently of each other in those periods between the data exchange. Therefore, they can be executed concurrently in order to increase the model performance.

For the components in this scenario, components need to be specified with a fixed time step. This information is required for the scheduler in order to optimise the scheduling. In case the modeller wants to minimise initialising and suspending of components each time step, the component must be implemented in a way that the component can proceed a single time step like C_2 or an interval of time steps like C_1. However, if a component proceeds via a series of time steps, it is still necessary to store the states of each time step in order to give other components access to the state values within this series.

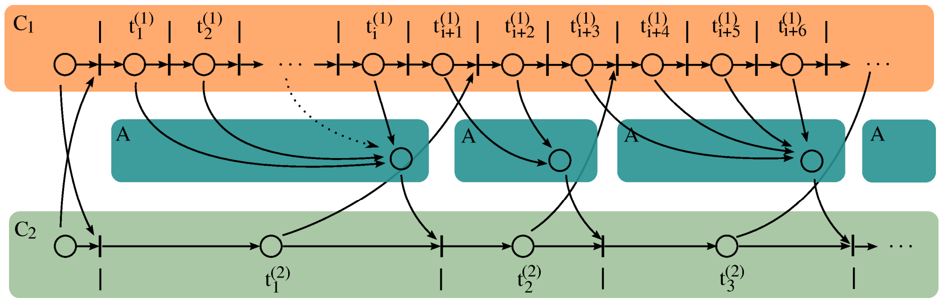

Figure 2: Temporal control flow between interacting components where component C_2 changes the time step lengths. The scheduling needs to update the interval used by the adapter and to notify C_1 of the the advanced availability of data from C_2.

Incorporating components with variable time step

While a fixed time step is used in most models, not all situations are covered with the scheduling approach outlined in the previous section. We will now consider 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 modelling plant growth with low granularity in the summer season and higher in the winter. Variations in time steps can occur in two ways: known before model execution, or known during runtime.

First, we consider the case when variation in the time steps is known before the model execution. This situation is shown in Figure 2. In terms of component interactions this situation is identical to the one described in the previous section, C_2 requires aggregated values from C_1. C_2 changes its time step after time step $t_1^{(2)}$ to a shorter duration. As a consequence, the scheduling needs to adapt the interval used by the adapter as well as the data exchange moment between C_1 and C_2. As the variation in the time steps of the component is known at the design stage, the scheduling of components can be derived before the execution of the model. Therefore, previously described approaches to optimise runtime series of components also apply in this case.

Second, a model component can vary its time step duration during model runtime, the duration of the time step is determined at the start of the time step because it may depend on input data. In this case, it is not possible to derive an execution schedule for the total model runtime in advance. Therefore, it is compulsory to evaluate the scheduling of components at runtime.

For the scheduler, the two situations call for a different approach to derive the schedules. For known time step variations, the calculation of the schedule is similar to the one described in the previous section, and schedule can be calculated completely before execution.

For unknown time steps at runtime the calculation of the schedule can only be carried out for a limited time in advance. The components can be scheduled and executed until the next synchronisation moment, after that the next synchronisation moment is determined by one of the variable time step components. Until then, scheduling and execution can be applied again. This alternating scheme continues until the end of the simulation.

For modelling situations with variable time steps, two additional types of components are required. One component type with variable, but known time steps before the model run. This type of components needs to provide a list of time steps to the scheduler instead of a fixed interval. The second type of component holds time steps that are variable and unknown before the model run. Therefore, these component types must communicate the end of the current time step to the scheduler.

Incorporation of components with unknown start and end time

The implications of the existence of components with flexible time steps on the prediction of the execution scheme was already discussed in the previous section. Next, we consider the case where even less knowledge about the component lifetimes is available. This situation imposes a more flexible scheduling scheme, but allows for less options to optimise the execution scheme. Here, we consider the situation in which an unknown, limited number of components can appear and disappear during model runtime (Figure 3). An example is a forest model where a field based groundwater component is coupled to a number of components representing individual trees. The trees have their own confined lifetime, can spawn offspring, and interact by water extraction with the groundwater component. Before the simulation run, the functions and parameters describing these processes are known. Also known is the initial population with random age values, while the variation in the number of trees is unknown before the model run.

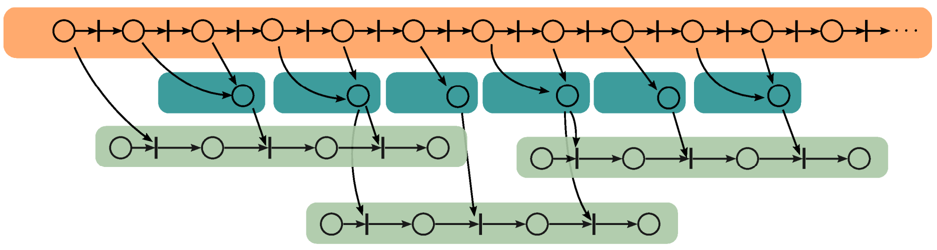

Figure 3: Components with limited life time interact with a component with continuous time steps. The scheduling needs to update the interval used by the adapter and to keep track of the number of individual components. Also, aggregated values are used as initial values for the individual components.

.

To organise the scheduling of components with unknown lifetime, the scheduling needs to be arranged in a flexible way regarding the existing situation present in the model, that is the number of participating components. Therefore, the scheduler can only consider a short timeframe in advance. In order to achieve this, the components need to be registered at the scheduler as well as the adapter used to communicate with other components. The registration allows the scheduler to generate dynamic lists that hold the properties of those individual components. These lists are continuously updated regarding for 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 at model runtime.

The interactive components must hold their start time, end time and time step that determine the temporal properties of such a model component. Furthermore, a method needs to be implemented to determine the action at a spawn moment such as cloning the current component type as the establishment of a new tree individual, or the launching of another component type instance such as an event-triggered avalanche component. As the number of components is unknown at runtime an increased effort in the communication between the components and the scheduler is required. For example, a tree model component calls the scheduler in order to obtain the neighbouring trees at specific moments in time. Also, a component needs to obtain information about the scheduler in order to forward this information to the spawned component allowing its self-registration at the scheduler. In order to spawn components at individual moments during the model run in a flexible way it is recommended to construct autonomous components that can be activated by and communicate with a central control instance.

Control of communication and execution flow

The organisation of component execution and data transfer between components requires information transfer between components about when and what information has to be transferred. The ordered execution of individual processes is required in other domains as well and can be found for example in the process scheduling of operating systems (e.g., Torrey et al., 2007) or production line planning executed in operations research (e.g., Koomsap et al., 2005).

In OpenMI (Moore and Tindall, 2005), the inter-component communication, that is notifying one or more components about current status and requests for data, is done by a pull-based approach. This means a component requests information and thereby initiates the progress of the delivering component. The opposite approach is to keep a centralised instance organising the execution and communication of components by maintaining a shared timeline. This client-server relationship is for instance used in the Tarsier framework (Watson and Rahman, 2004).

Here, we follow the approach of a central instance organising the execution of components by maintaining a shared timeline. This client-server approach offers higher flexibility in the execution schemes over the pull-based method such as concurrent scheduling of components.

SCHEDULE GENERATION AND COMPONENT EXECUTION

Based on the different scenarios described in the previous sections, we now present the framework for schedule generation, and component execution. The three layer design of the framework separated in coupled component description, schedule generation and component execution is shown in Figure 4. With this approach, we can seamlessly provide both flexible and optimised execution schemes within one control environment.

The model builder describes the coupled model in layer one, the model definition layer. Here, the different component types, their simulation horizon, the component interactions as well as the adapter are specified with the help of a scripted user interface.

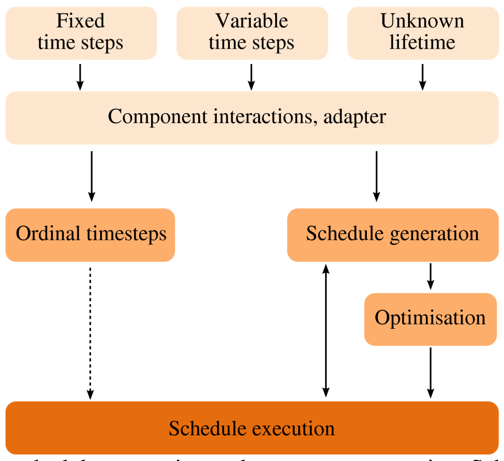

Figure 4: Stages of the schedule generation and component execution. Schedules with fixed time steps can be optimised regarding the runtimes. With unknown time step lengths, a continuous interaction between scheduler and executing component is required.

This information is passed to layer two, the schedule generation layer. Based on the component types the schedule generation and interaction with the schedule execution layer is determined. For fixed temporal situations, the schedule is derived for the total simulation time and calculated in advance. Afterwards, component interval simulation times can be optimised for the complete schedule. Thereafter, the schedule can be either written to disk or passed to the component executing layer.

For situations with variable time steps, the schedule generation layer needs to communicate with the component executing layer. The scheduler will calculate an appropriate simulation time interval in advance, which is then forwarded to the execution layer. Afterwards, the components will be executed for that simulation time interval. Once all components finish that interval, the execution layer requests the subsequent schedule interval from the schedule generation layer. Scheduling and executing layer will alternate this procedure until the end of simulation time.

The third layer holds the part of the framework that arranges the component execution, fed either with a precalculated schedule or closely interacting with the schedule generator. By separating the schedule generation and execution we can offer a higher degree of flexibility in the component execution. Close interaction of the scheduler component with the generator part allows for dynamic modelling schemes. Accepting precalculated schedules provides several additional execution options. Models can be rerun with a predefined order of component execution. The modeller can decide as well based for example on hardware specification or component parallelism if components in a model run should be executed concurrently or sequentially. Moreover, the schedule generating component with its default Gregorian calendar can be bypassed and the executing component can be fed with user made schedules. Therefore, model schemes with time horizons such as 500000 years and time steps of 10000 years can be executed.

SOFTWARE IMPLEMENTATION

Currently, we are developing a software prototype that implements the concepts of the scheduling and execution framework described in the previous section. We are using the scripting language Python as implementation language for component development and coupling. The Python language is easy to use by non-software developers, offers rich functionality for scientific computing and visualisation (e.g., SciPy, 2011: RPy, 2011: Karssenberg et al., 2007). Also, interoperability between Python and system programming languages such as Fortran or C++ is straightforward to accomplish (e.g., Langtangen, 2007).

The developer of an integrated model constructs components and adapter with the help of Python classes analogue to the approach introduced in Karssenberg et al. (2010). Figure 5 shows a source code example for a component modelling the groundwater flow process. The software framework provides a set of base classes for process components and adapter. These classes provide generic functionality such as input and output methods. In addition, they state requirements that need to be implemented by the component developer such as the process descriptions within a time step. By complying to the framework classes, components can always be coupled to other components. The framework classes can also be utilized as a wrapper by forwarding input and output data to an external system, for example a command line application.

```python
from components.fixedTimestep import *

class Groundwater(FixedTimeStep):
  def __init__(self, IoInterface, startTime, endTime, deltaT):
    # binding external input and output specification to
    # read and report methods
    self.initialiseInputOutput(IoInterface)

  def init(self):
    # setting starting values

  def runTimestep(self):
    # obtaining external data
    value = self.read("percolation")
    # process descriptions for one timestep, omitted
    ...
    # making state variable available for other components
    self.write("seepage")
```

Figure 5: Python code snippet showing the implementation of the groundwater component. Generic functionality such as executing one or several time steps is derived from parent classes, here `FixedTimestep`. Process specific descriptions need to be implemented by the model developer. Input and output interfaces specified by the file `IoInterface` are used by the `read` method to obtain values from another component during a model run.

These components are coupled as shown in the script of Figure 6. The components are initialised with start and end time as well as the time step duration, here in days. The component interface is described in a separate XML file holding the variable specific information such as the data type for each input and output data. Below the component initialisation (line 7 to 9), the `addComponent` and `addAdapter` calls build up the shared timeline of the coupled model. The execution order of components and adapter is calculated by the `generateSchedule` call. Finally, components and adapter are executed according to the schedule.

```python
 1 from scheduler import *
 2 # components/adaptor described in separate Python classes
 3 from groundwater, percolation, percToGroundwater import *
 4
 5 # instanciate scheduler, process and adapter components
 6 sched = Scheduler()
 7 swp = Percolation("swp.xml", "1980-01-01", "1980-12-31", "1")
 8 gwater = Groundwater("gw.xml","1980-01-01", "1980-12-31", "30")
 9 percToGroundwater = PercToGroundwater("p2gw.xml")
10
11 # introduce components and adapter
12 sched.addComponent(swp)
13 sched.addComponent(gwater)
14 sched.addAdapter(percToGroundwater)
15
16 # generate schedule and execute:
17 sched.execute(sched.generateSchedule())
```

Figure 6: Code snippet showing the implementation of the coupled model introduced in Section 2.1. First, component and adapter with properties such as input/output interfaces and temporal specifications are instantiated. These are passed to the schedule generator, which determines the order of component execution.

CONCLUSION AND FUTURE WORK

The scheduling framework and software prototype presented in this paper allow the construction of model components and their coupling in consideration of different temporal resolutions. By utilising a high-level scripting language for the model description we envision an easier construction process of integrated models for scientific research.

The framework assumes components to progress forward in time. Processes requiring close interaction within a time step, as for example in solving differential equations between two components, can therefore not be represented adequately. In the current state, the framework exclusively addresses single deterministic model executions, while environmental processes are predominantly stochastic. Therefore, further research will include the extension of the framework by execution schemes such as Monte Carlo (e.g., Doucet et al., 2001) and Particle Filtering (e.g., Moradkhani et al., 2005) in order to estimate uncertainty in coupled models.

Accompanying this task, methods for extraction of component results for analysis and visualisation need to be enhanced. Hence, the formalised semantic description (e.g., Rizzoli et al., 2008: Villa et al., 2009) of the software framework will be extended. As a potential result, interoperability with components complying to other frameworks such as OpenMI, or web-based data provider (e.g., Goodchild et al., 2007) can be facilitated.

REFERENCES

Argent, R., Perraud, J.-M., Rahman, J., Grayson, R., Podger, G., 2009. A new approach to water quality modelling and environmental decision support systems. Environmental Modelling & Software 24 (7), 809–818.

Argent, R. M., 2004. An overview of model integration for environmental applications–components, frameworks and semantics. Environmental Modelling & Software 19 (3), 219–234.

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
.

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

Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., da Silva, A., 2005. Design and implementation of components in the Earth System Modeling Framework. International Journal of High Performance Computing Applications 19 (3), 341–350.

de Kok, J.-L., Engelen, G., Maes, J., 2010. Towards Model Component Reuse for the Design of Simulation Models – A Case Study for ICZM. In: Swayne, D. A., Yang, W., Voinov, A. A., Rizzoli, A., Filatova, T. (Eds.), International Congress on Environmental Modelling and Software Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada. International Environmental Modelling and Software Society (iEMSs).

Donatelli, M., Rizzoli, A., 2008. A design for framework-independent model components of biophysical systems. In: Sànchez-Marrè, M., Béjar, J., Comas, J., Rizzoli, A., Guariso, G. (Eds.), Integrating Sciences and Information Technology for Environmental Assessment and Decision Making. iEMSs 2008: International Congress on Environmental Modelling and Software, pp. 727–734.

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

Engelen, G., 2004. Models in Policy Formulation and Assessment: The WadBOS Decision Support System. In: Environmental Modelling: Finding Simplicity in Complexity. John Wiley & Sons Ltd, pp. 257–271.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: elements of reusable object-oriented software. Addison Wesley.

Goodchild, M. F., Fu, P., Rich, P., 2007. Sharing Geographic Information: An Assessment of the Geospatial One-Stop. Annals of the Association of American Geographers 97 (2), 250–266.

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

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

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

Karssenberg, D., de Jong, K., van der Kwast, J., 2007. Modelling landscape dynamics with Python. International Journal of Geographical Information Science 21 (5), 483–495.

Karssenberg, D., Schmitz, O., Salamon, P., de Jong, K., Bierkens, M. F., 2010. A software framework for construction of process-based stochastic spatio-temporal models and data assimilation. Environmental Modelling & Software 25 (4), 489–502.

Koomsap, P., Shaikh, N. I., Prabhu, V. V., 2005. Integrated process control and condition-based maintenance scheduler for distributed manufacturing control systems. International Journal of Production Research 43 (8), 1625–1641.

Langtangen, H. P., 2007. Python Scripting for Computational Science, 3rd Edition. Springer.

Letcher, R., Croke, B., Jakeman, A., 2007. Integrated assessment modelling for water resource allocation and management: A generalised conceptual framework. Environmental Modelling & Software 22 (5), 733–742.

Liu, J., Peng, C., Dang, Q., Apps, M., Jiang, H., 2002. A component object model strategy for reusing ecosystem models. Computers and Electronics in Agriculture 35 (1), 17–33.

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

Moradkhani, H., Hsu, K.-L., Gupta, H., Sorooshian, S., 2005. Uncertainty assessment of hydrologic model states and parameters: Sequential data assimilation using the particle filter. Water Resour. Res. 41, W05012.

North, M., Collier, N., Vos, J., 2006. Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Transactions on Modeling and Computer Simulation 16 (1), 1–25.

Parker, P., Letcher, R., Jakeman, A., Beck, M. B., Harris, G., Argent, R. M., Hare, M., Pahl-Wostl, C., Voinov, A., Janssen, M., Sullivan, P., Scoccimarro, M., Friend, A., Sonnenshein, M., Barker, D., Matejicek, L., Odulaja, D., Deadman, P., Lim, K., Larocque, G., Tarikhi, P., Fletcher, C., Put, A., Maxwell, T., Charles, A., Breeze, H., Nakatani, N., Mudgal, S., Naito, W., Osidele, O., Eriksson, I., Kautsky, U., Kautsky, E., Naeslund, B., Kumblad, L., Park, R., Maltagliati, S., Girardin, P., Rizzoli, A., Mauriello, D., Hoch, R., Pelletier, D., Reilly, J., Olafsdottir, R., Bin, S., 2002. Progress in integrated assessment and modelling. Environmental Modelling & Software 17 (3), 209–217.

Pullar, D., 2003. Simulation Modelling Applied To Runoff Modelling Using MapScript. Transactions in GIS 7 (2), 267–283.

Rivington, M., Matthews, K., Bellocchi, G., Buchan, K., Stöckle, C., Donatelli, M., 2007. An integrated assessment approach to conduct analyses of climate change impacts on whole-farm systems. Environmental Modelling & Software 22 (2), 202–210.

Rizzoli, A. E., Donatelli, M., Athanasiadis, I. N., Villa, F., Huber, D., 2008. Semantic links in integrated modelling frameworks. Mathematics and Computers in Simulation 78 (2-3), 412–423.

Rotmans, J., 1990. IMAGE: an integrated model to assess the greenhouse effect. Kluwer.

RPy, 2011. Python interface to the R Programming Language. http://rpy.sourceforge.net/

SciPy, 2011. Scientific Tools for Python. http://www.scipy.org/

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

Szyperski, C., 2002. Component Software: Beyond Object-Oriented Programming, 2nd Edition. Addison Wesley.

Torrey, L. A., Coleman, J., Miller, B. P., 2007. A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler. Software: Practice and Experience 37 (4), 347–364.

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

Villa, F., Costanza, R., 2000. Design of multi-paradigm integrating modelling tools for ecological research. Environmental Modelling & Software 15 (2), 169–177.

Voinov, A., Fitz, C., Boumans, R., Costanza, R., 2004. Modular ecosystem modeling. Environmental Modelling & Software 19 (3), 285–304.

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

Wesseling, C., Karssenberg, D., van Deursen, W., Burrough, P., 1996. Integrating dynamic environmental models in GIS: the development of a Dynamic Modelling language. Transactions in GIS 1, 40–48.