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 Department of Physical Geography, Utrecht University, The Netherlands (o.schmitz@geo.uu.nl)

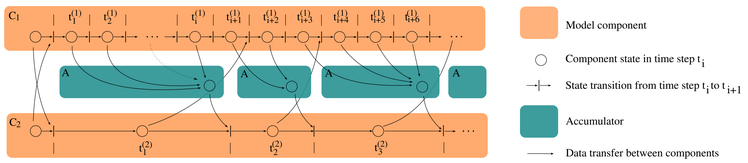
2 Flemish Institute for Technological Research (VITO), Belgium

Motivation

The construction of integrated models holding components of environmental, social and economic systems is relevant for scientific research and decision support. Software tools supporting the construction of these models exist, but they often impose extensive knowledge of system programming languages and software engineering on the user.

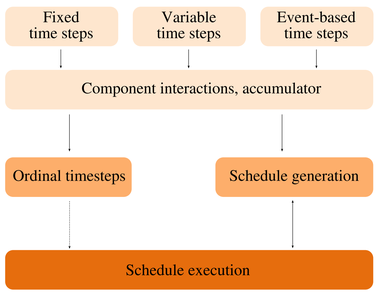
Objective

The objective of this research is the development of an environment for model component construction, coupling and assessment tailored to explorative model construction.



The framework eases the development of models consisting of several components with different time steps. State variables can be transferred between components either directly, or aggregated by an interconnected accumulator.

Software architecture



The user develops components with flexible time steps and accumulators with the help of base classes provided by the framework. The temporal discretisation and interactions between components determine the execution order during the model run. Base classes and model execution are provided as modules for the high-level programming language Python.

Example model

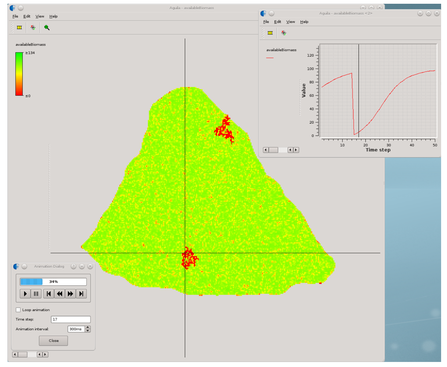
We apply the framework to a two-component model with bidirectional interaction and each different time steps. The coupled model consists of a component simulating biomass growth processes (Dakos, 2009) on a fixed yearly timestep, and a component simulating fire spread (Karafyllidis, 1997) on a variable time step.

```
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
5    self.biomass = self.readmap("initialBiomass")
6    self.addOutputValue(self.biomass)
7    self.burned = None
8    self.addInputValue(self.burned)
9    self.numberOfNeighbours = window4total(spatial(scalar(1)))
10
11 def runTimestep(self):
12   self.biomass = ifthenelse(self.burned, 0.1, self.biomass)
13   growth = 0.20 * self.biomass * (1 - self.biomass / 100.0) - ↩
       (self.biomass**2 / (self.biomass**2 + 1))
14   sumBiomassOfNeighbours = window4total(self.biomass)
15   diffusion = 0.01 * (sumBiomassOfNeighbours - ↩
       self.numberOfNeighbours * self.biomass)
16   growth = growth + diffusion
17   self.biomass = self.biomass + growth
18   self.report(self.biomass, "availableBiomass")
```

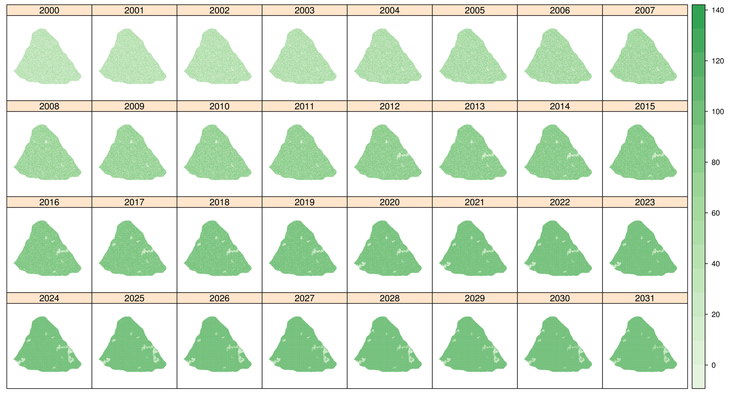
The listing above shows the biomass component with framework specific functionality of temporal discretisation as well as input and output specification. Spatial operations for the process description are realised with PCRaster operations (Karssenberg, 2007).

```
1  biomassFire = CoupledModel()
2  biomass = Biomass()
3  fire = Fire()
4  fireAcc = FireAccumulator()
5
6  biomassFire.add(biomass)
7  biomassFire.add(fire)
8  biomassFire.add(fireAcc)
9
10 biomassFire.link(biomass, "biomass", fire, "availableBiomass")
11 biomassFire.link(fire, "burnedArea", fireAcc, "burnedArea")
12 biomassFire.link(fireAcc, "burnedAcc", biomass, "burned")
13
14 scheduler.SingleRun(biomassFire).run()
```

Listing showing the construction and execution of the coupled model composed of model components and an accumulator. The data exchange is specified by links between the state variables of the components and accumulator, respectively.



Screenshot displaying the biomass content within the catchment for the year 2017. The timeseries indicates the biomass decrease caused by a fire in the selected location. Cell values and times can be queried interactively.



Output series of the biomass component for a simulation run starting from a random biomass distribution. Bright green spots indicate areas burned in the previous year.

Conclusions

The framework provided as Python module in combination with existing Python libraries allows straightforward development of spatio-temporal models with variable temporal discretisations. Future research will focus on the integration of advanced analysis schemes and the extension of formalised component descriptions.

References

Dakos, V., van Nes, E. H., Donangelo, R., Fort, H., Scheffer, M., 2009. Spatial correlation as leading indicator of catastrophic shifts. Theoretical Ecology 3 (3), 163–174.

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

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.