A framework for integrated, multi-scale model construction and uncertainty assessment

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

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

2 Flemish Institute for Technological Research (VITO), Belgium

Motivation and objective

The construction of self-contained modules with standardised interfaces leads to the development of more generic and reusable component models. However, required inputs can be obtained from different sources such as static data read from disk (e.g. by readmap("runoff")) or as output from other components generated at model runtime (e.g. by runoff.get()). Different sources require a different input request syntax within a state transition function and therefore a dependency between the implementation of the state transition function and the input interface is introduced.

We aim to provide a uniform specification of the input requests within a state transition function independent of the source type. We propose the function object notation as means to specify input requirements and apply this approach to an integrated modelling framework implemented in Python. The used request-reply execution supports for Monte Carlo simulations.

Building blocks and modelling framework

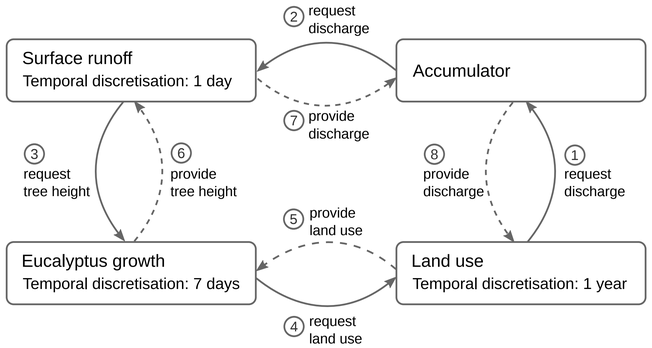
The modelling framework contains template classes allowing for dynamic modelling and Monte Carlo simulations. The modeller can use map algebra operations provided by the PCRaster Python module [1, 2] to implement stochastic spatial processes.

A function notation of input requests implicitly defines the execution order of component models. Called components can execute several time steps until the time step matches the one of the requesting component. Monte Carlo simulations of integrated models are executed according to the following scheme:

```
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
```

Example model

We apply the framework to build an integrated model with components modelling land use change, surface runoff and eucalyptus tree growth. The components use different time steps, discretisation differences can be bridged with the help of accumulators [3]. The numbers indicate the order of requests and replies:

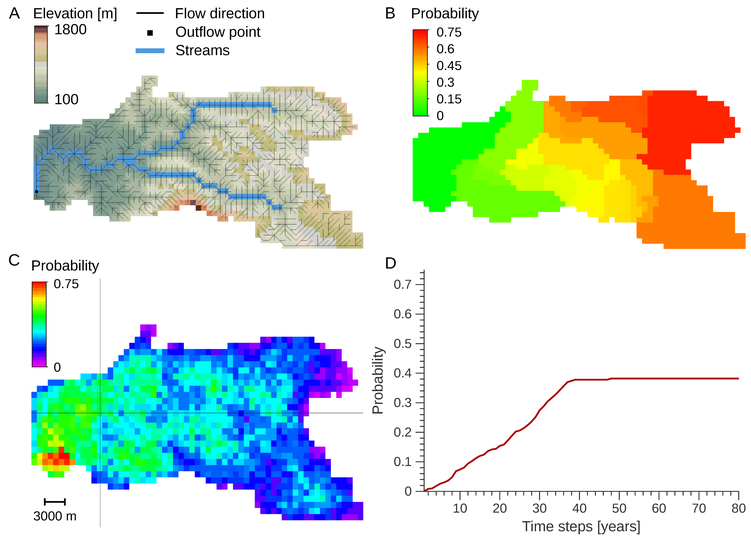


The following listing shows the implementation of the land use change component and its instantiation within the modelling framework. The component initiates every year an input request to the runoff accumulator. Each component can implement its own postmcloop with operations calculating ensemble statistics.

```
1  class Landuse(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.tot_runoff = Input(TotalRunoff(start, end, timedelta(days=1), cloneMap))
7
8    def premcloop(self):
9      self.yearlyRunoffRequired = self.readmap("requiredRunoff")
10     self.mainStreams = self.readmap("streams")
11     self.ldd = lddcreate("dem.map", 1e31, 1e31, 1e31, 1e31)
12
13   def initial(self):
14     self.trees = spatial(boolean(0))
15     self.export(self.trees, "trees")
16
17   def dynamic(self):
18     total_runoff = self.tot_runoff(self.current_time_step())
19     runoffTooLow = self.mainStreams & (total_runoff.totRunoff < self.yearlyRunoffRequired)
20     noNewTrees = catchment(self.ldd, runoffTooLow)
21     suitabilityNeighbourhood = ifthenelse(window4total(scalar(self.trees)) > 0.5, scalar(1), 0)
22     suitabilityRandom = ifthen(pcrnot(self.trees), uniform(1))
23     suitability = ifthen(pcrnot(noNewTrees), (suitabilityNeighbourhood + suitabilityRandom) / 2.0)
24     suitSort = order(0.0 - suitability)
25     self.trees = pcror(self.trees, cover(suitSort < 11, 0))
26     self.export(self.trees, "trees")
27
28   def postmcloop(self):
29     mc_probability("trees", self.sampleNumbers(), self.timeSteps())
30
31 model = Landuse(datetime(2010, 1, 1), datetime(2100, 1, 1), timedelta(days=365), "clone.map")
32 dynModel = DynamicFramework(model)
33 mcFrw = MonteCarloFramework(dynModel, nrRealisations=500).run()
```

The Input class returns a function object that is assigned to the tot_runoff variable (line 6) at initialisation of the component model. While executing the dynamic section at runtime, the function call (line 18) initiates the execution of the invoked component until the current time step.
The process implementation (the dynamic section) remains the same irrespective of input data obtained from disk or external components.

Outputs



Results can be visualised for each component, timestep and cell. The figure shows the catchment (A), the probability of cells being excluded from planting due to water scarcity in 2045 (B) and the probability of trees present in a cell in 2100 (C). Timeseries can be displayed interactively for each cell (D).

Conclusions

The function notation syntax for input requirements decouples the implementation of the state transition function and the interface specification, and therefore increases the generic design of component models. Model builders therefore can easier select and assess alternative model compositions.

References

[1] http://www.pcraster.eu
[2] 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.
[3] Schmitz, O., Salvadore, E., Poelmans, L., van der Kwast, J., Karssenberg, D., 2014. A framework to resolve spatio-temporal misalignment in component-based modelling. Journal of Hydroinformatics 16(4), 850-871.