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

Oliver Schmitz¹, Jean-Luc de Kok², Kor de Jong¹ and Derek Karssenberg¹

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 component models. However, required inputs can be 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


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 postloop with operations calculating statistic values.

```
1 class Landuse(DynamicModel, MonteCarloModel):
2     def __init__(self, start, end, delta, cloneMap):
3         dynModel = DynamicFramework(model)
4         mcFrw = MonteCarloFramework(dynModel, nrRealisations=500).run()
5         return mcFrw
6
7     def postloop(self):
8         # request trees from upstream
9         self.trees = self.mcFrw("trees", self.sampleNumbers(), self.timeSteps())
10        # eucalyptus growth
11        suitSort = order(0.0 - suitability)
12        self.trees = pcror(self.trees, cover(suitSort < 11, 0))
13        # land use change
14        self.current_time_step() = self.current_time_step() + 1
15        self.landuse = self.mcFrw("land use", self.sampleNumbers(), self.timeSteps())
16        self.current_time_step() = self.current_time_step() + 1
17        self.landuse = self.mcFrw("land use", self.sampleNumbers(), self.timeSteps())
18        # surface runoff
19        total_runoff = self.mcFrw("total runoff")
20        self.runoff = total_runoff & (total_runoff.totRunoff < self.yearlyRunoffRequired)
21        self.runoff = self.runoff & self.landuse & self.accumulator
22        # eucalyptus growth
23        self.trees = pcror(self.trees, cover(suitSort < 11, 0))
24        # export
25        self.export(self.trees)

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).

Outputs

A Elevation [m] Flow direction Outflow point Streams
B Probability
C Probability
D Probability

References