Creating your own LUCCME component

A developer's guide

Version 3.1 | September 2017

Creating your own LUCCME component

A developer's guide

Version 3.0 | December 2016

Lead authors: LuccME Team

Contents

1 Introducing the LuccME framework

1.1 What is LuccME

LuccME is an open source framework for spatially explicit Land Use and Cover Change (LUCC) modeling developed by the Earth System Science Center (CCST) and collaborators, built on top of the Terra-ME (Carneiro et al., 2013), as an extension. Using LuccME the modeler can easily create deforestation, agricultural expansion, desertification, forest degradation, urban sprawl spatial models and other land use change models at different scales and areas of study, combining existing model components and/or creating new ones.

1.2 The philosophy of LuccME

There are many different types of LUCC models which can be classified according to model goal, scale, technical approach or underlying theory. In spite this diversity of modeling approaches, a common structure can be identified in several spatially explicit models (Verburg et al., 2006; Eastman et al, 2005), as Figure 1 illustrates. In general, three main components can be identified: *Demand* – responsible by the calculation of the magnitude or quantity of change; *Potential* – responsible by the calculation of the suitability or propensity of change for each cell; *Allocation* – responsible by the spatial distribution of changes based on land demand and potential of change of each cell. These components are organized in a top-down manner, in which a demand for land is spatially allocated according to cell suitability. Several well known LUCC models follow this structure, including CLUE family (Veldkamp and Fresco, 1996; Verbug et al., 1999; Verburg et al., 2002), Dinamica EGO (Soares-Filho et al. 2002), and GEOMOD (Pontius et al., 2001), using a range of different approaches and techniques for the three components (Eastman et al., 2009; Lesschen et al., 2007). However, these models are implemented in different computing platforms, their code is in general not open, and they cannot be easily modified or combined.

In this sense, LuccME allows the construction of new models, combining elements of *Demand*, *Potential* and *Allocation* components, designed according to the original idea of the main models found in the literature. However, a distinguishing feature of LuccME is its modular set-up. This

allows tailoring of the model structure to the user needs. In LuccME, the components can be chosen and modified according to the needs of a given application and scale of analysis, and easily parameterized through a simple user interface. LuccME is built on top of TerraME, a general programming environment for spatial dynamical modeling, designed to support models in several domains, including hydrology, biodiversity, land cover change, and many others. Its modeling language has in-built functions that make it easier to develop multi-scale and multi-paradigm models for environmental applications. TerraME provides an interface to TerraLib geographical library (Camara et al., 2008), allowing models direct access to geospatial data. LuccME, TerraME and Terralib are technological products developed by INPE and its partners.

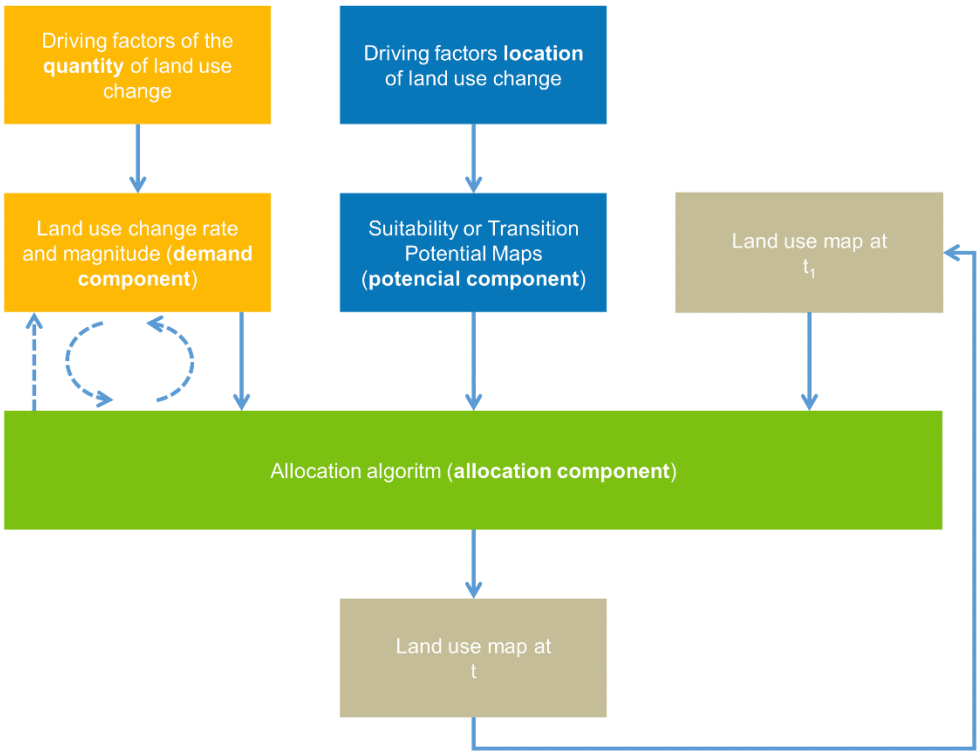

Figure 1 - Generic structure of the main spatially explicitly LUCC Models available in the literature (adapted from Verburg et al. 2006).

Besides providing an easy way to use and create new LUCC models, LuccME was designed in order to support the development of LUCC models which can be integrated to more complex multi-scale environmental models. In this sense, models created using LuccME are encapsulated in Environments (a concept proposed by Carneiro (2006) and a core concept in Terra-ME), so later they may be easily coupled to other models, for instance Earth System models and/or multi-scale LUCC models.

2 Building a component for LuccME

2.1 Introduction

In order to build a new component for LUCCME the developer must know its architecture and follow the defined interface *(data that must be present on the output component)*.

The Figure 2 shows the LUCCME basic architecture:

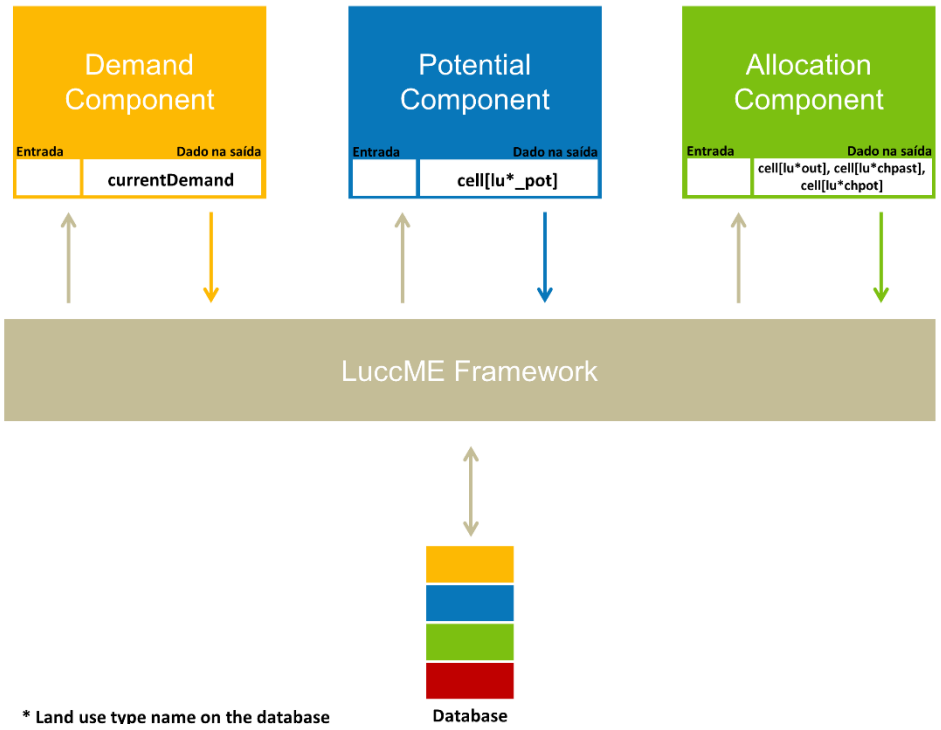

Figure 2 - LUCCME basic architecture

Every component must have at least two methods, verify and execute, but the developer can create as many as it necessary.

The framework first call the verify method of a component and after the execute one.

The verify method must check all the inputs of the component, such as: if all the inputs are declared on the configuration file *(if it is not optional)*; the validation of an input within database or range of values.

The execute method implements the component rules to run itself.

Below shows an empty component example:

```
--- Empty component example
-- @arg component An instance of the component itself (an object, with all the data).
-- @arg component.execute Handles with the execution method of an EmptyComponent component.
-- @arg component.verify Handles with the verify method of an EmptyComponent component.
-- @return The modified component.
-- @usage emptyComponent = EmptyComponent {}
function EmptyComponent(component)
	-- Handles with the rules of the component execution.
	-- @arg self A EmptyComponent component.
	-- @arg event A representation of a time instant when the simulation engine must execute.
	-- @arg luccMeModel A container that encapsulates space, time, behavior, and other
	-- environments.
	-- @usage self.emptyComponent:execute(event, model)
	component.execute = function(self, event, luccMEModel)
	end

	-- Handles with the parameters verification.
	-- @arg self A EmptyComponent component.
	-- @arg event A representation of a time instant when the simulation engine must execute.
	-- @arg luccMeModel A container that encapsulates space, time, behavior, and other
	-- environments.
	-- @usage self.emptyComponent:verify(event, model)
	component.verify = function(self, event, luccMEModel)
	end

	return component
end
```

Besides of the return of a component it is always the modified component, each type of component have its own peculiarities, a set of data that must be present on the modified component, and that will be shown on the next sessions.

2.2 Demand Component

The demand component is the first to be called on the framework, it handles with the demand of the land use types yearly for the simulation period of time.

The input of a demand component is usually the demand of the land use types for each year of simulation *(annual demand)*.

The output of a demand component must have the currentDemand table, its contains the demand generated for each time of simulation for each land use type used on the application, based on the rules implemented on the execute method.

2.3 Potential Component

The potential component is the second to be called on the framework, it handles with the potential of the land use types yearly for the simulation period of time.

The input of a potential component is the data relevant to calculate the potential, it does not have any rule, it changes according to the component needs.

The output of a potential component must have the potential for each land use types for each cell yearly for the period of simulation, following this patter: cell[<landUseTypeName>_pot], where <landUseTypeName> is the name of the land use type used on the application.

For example:

> Using the land use types: forest, deforest and others; the potential must have on its output a cell[forest_pot], cell[deforest_pot] and cell[other_pot].

2.4 Allocation Component

The allocation component is the third and last to be called on the framework, it handles with the allocation of the land use types yearly for the simulation period of time.

The input of an allocation component is the data relevant to calculate the allocation, usually it uses the demand and potential, but it does not have any rule, it changes according to the component needs.

The output of an allocation component must have the allocation for each land use types for each cell yearly for the period of simulation, following this patter: cell[<landUseTypeName>_out] and the changes between the start-time and allocation year: cell[<landUseTypeName>_chtot], where <landUseTypeName> is the name of the land use type used on the application, and the changes between the last year and allocation year: cell[<landUseTypeName>_chtot].

For example:

> Using the land use types: forest, deforest and others; the allocation must have on its output:
>
> cell[forest_out], cell[deforest_out], cell[other_out];
>
> cell[forest_chtot], cell[deforest_chtot] and cell[other_chtot];
>
> cell[forest_chpast], cell[deforest_chpast] and cell[other_chpast];