International Congress on Environmental Modelling and Software

7th International Congress on Environmental Modelling and Software - San Diego, California, USA - June 2014

Jun 16th, 9:00 AM - 10:20 AM

An Environmental Modeling Language for Agents and Fields

Kor de Jong
Utrecht University, k.dejong1@uu.nl

Merijn de Bakker
Utrecht University, m.p.debakker@uu.nl

Derek Karssenberg
Utrecht University, d.karssenberg@uu.nl

International Environmental Modelling and Software Society (iEMSs)
7th Intl. Congress on Env. Modelling and Software, San Diego, CA, USA
Daniel P. Ames, Nigel W.T. Quinn and Andrea E. Rizzoli (Eds.)
http://www.iemss.org/society/index.php/iemss-2014-proceedings

An Environmental Modeling Language for Agents and Fields

Kor de Jong [a] **, Merijn de Bakker** [a] **and Derek Karssenberg** [a]

[a] *Utrecht University, Faculty of Geosciences, Department of Physical Geography, PO Box 80.115, 3508 TC, Utrecht, Netherlands (k.dejong1@uu.nl, m.p.debakker@uu.nl, d.karssenberg@uu.nl)*

Abstract: Environmental modeling involves manipulating environmental attributes represented in software by agents, fields or both, but most modeling environments are designed to be especially useful for either agent-based or field-based modeling. Agent-based and field-based modeling environments have different properties with respect to their ease of use and how well both agents and fields can be represented and manipulated. Most agent-based modeling environments require the modeler to use a general purpose object oriented programming language like Java to express models, while field-based modeling environments often implement a high level, domain specific imperative language based on map algebra, or extent a general purpose scripting language. Because of this, field-based models are more easily defined by domain experts than agent-based models. On the other hand, because a lower level language is used, agent-based modeling environments are more easily extended by missing functionality, like support for fields, while in general, field-based modeling environments lack support for defining agents. We are working on a new environmental modeling environment which is designed from the ground up for manipulating both agents and fields. Our goal is to combine the advantages of current agent-based and field-based modeling environments. For this, a conceptual data model is developed which is capable of storing both agent and field data, and which is a superset of the traditionally separate data models for agents and fields. Given this data model a high level domain specific imperative language is designed which enables domain experts to define combined agent- and field-based models.

Keywords: environmental modeling; agents; fields; data model; spatio-temporal data

1 INTRODUCTION

In environmental modeling, attributes about the state of the environment are manipulated. Each of these attributes represents a property of an environmental entity. Examples of such entities and their attributes are wolves with energy level and birth date attributes, regions with elevation and land-use attributes, and trees with biomass and species attributes. The entities modeled in an environmental model can be conceptualized as objects (or agents) and as continuous fields (Goodchild et al. [2007]). Agents are often defined as being spatially discrete, autonomous, modular and social (Macal and North [2009]), and can often be linked to real-world entities. Fields, on the other hand, are used to represent spatially continuous attributes.

Modeling environments are used to create environmental models and may offer support for visualizing, debugging, calibrating and optimizing models. A modeling language is used to define environmental models and can be a textual or graphical language. Here we focus on textual languages. Modeling languages range from higher level domain specific languages like NetLogo (Wilensky [1999]) to lower level general purpose languages like FORTRAN. One can assume that domain specific languages are better suited for domain experts without experience in software development. While developing concepts and software for environmental modeling we especially target this group of modelers.

Agent-based and field-based modeling environments have different properties with respect to their ease of use and how well both agents and fields can be represented and manipulated. Most agent-based modeling environments require the modeler to use a general purpose programming language, like Java, to define models (Crooks and Castle [2006], Filatova et al. [2013], Wikipedia [2014]). This implies that modelers must either acquire the knowledge of programming a computer using such a language, or hire a capable person to translate the model rules into the language. But modelers are often domain specialists and may prefer not to have to learn a general purpose programming language. And hiring a programmer to build a model results in a black box (in the view of the modeler), which the modeller cannot easily update. An advantage of using a general purpose language as a modeling language is that adding missing functionality is relatively easy. For example, new modeling operations and even new data types can be defined and used in the model. Probably because of the similarities of model-agents and class instances in the object oriented programming paradigm, object orientation is often used to define agent-based models. In languages like C++, Java and Objective-C, the modeler defines agent classes, combining agent attributes and operations.

Field-based modeling environments often implement a high level, domain specific language based on map algebra (Wesseling et al. [1996], Neteler et al. [2012], Eastman [2009]), or extent a general purpose scripting language (Karssenberg et al. [2012], Zandbergen [2013]). An advantage of modeling environments implementing a domain specific language is that it is often usable by domain experts without knowledge about computer programming details, like memory and file management and error handling. These environments often lack support for working with agents and adding support for them is hampered by the fact that the high level data types and operations where not designed with agents in mind. Instead of object orientation used in agent based modeling, modeling languages for field-based models often use the simpler imperative (or procedural) programming paradigm, where the model contains all tasks to be done in the model in a sequence. There may be language constructs to define functions and organize model code in modules.

It is often necessary to model both agents and fields in a single model because there is a relation between an agent-attribute and a field-attribute. For example, in the wolves sheep case study presented later in this paper, there is a relation between the sheep agents and the grass field. Sheep may choose a direction to move based on the distribution of grass in the field, and the distribution of grass in the field is influenced by the location of the sheep. In general, in models where agents like individuals or herds influence their physical environment and are influenced by it, there is often a need to be able to combine agents and fields in a single model.

We are working on developing concepts for a new environmental modeling environment for manipulating both agent-based and field-based attributes. This paper describes the work done so far and focusses on the modeling language. The research question with respect to the modeling language is whether it is possible to design a high level imperative language for defining environmental models for manipulating agent-based and field-base attributes, much like field-based environmental modeling languages.

In the next sections we will first describe the data model with which we try to capture all kinds of environmental modeling data. After that we will describe the proposed language for manipulating the environmental entities and their attributes. To give an impression of how a complete agent-field model looks like in this new language, we will then look into a re-implementation of the well-known wolf sheep predation model from the NetLogo models library (Wilensky [1999]).

2 MODELING LANGUAGE

A modeling language is the user interface the environmental modeler uses to define models. A modeling language should enable modelers to efficiently express ideas about the environmental processes in a model. Ideally, the language should enable the modeler to turn ideas into running models as quickly as possible, and to even try out entirely different implementations. This requires a language that is easy to learn and use, and which exposes all the functionality the modeler needs, at the highest possible abstraction level. The modeler should not have to deal with resource management and error handling, for example.

The syntax of the language is an important property of a modeling language. It determines how the modeler instructs the modeling environment how to update the state of the environmental attributes. It also determines how easy it is for modelers to understand models defined earlier or defined by other modelers. Source code, including model code is read more often than it is written. When designing a modeling language it is important to take the scope of the language into account. The language is used to define environmental models, so it can do without constructs that are not relevant for the domain of environmental modeling.

The syntax of field-based environmental modeling languages meets the requirements for a simple, expressive modeling language. The drawback of these environments, as described above, is that these domain specific languages have been designed with a scope in mind that did not include agent-based attributes. Our goal is to determine whether it is possible to use the same syntax for combined agent-based and field-based attribute manipulation. This requires a new type of attributes to be added to the language. Besides information about the state of fields, it must be possible to represent information about the state of agents. For this we designed a new data model for representing all data that is relevant in environmental modeling.

2.1 Data model

A data model is a representation of reality in software. The data model underlying an environmental modeling environment determines the kind of data that can be manipulated in models. Field-based environmental modeling environments implement a data model for representing fields. Since we want to be able to model both agents and fields, we need a more generic data model, which is able to represent both agent and field data. It is also possible to use multiple data models, one for agents and one for fields, but this complicates the modeling environment's implementation and use. At a high abstraction level it often does not matter to the modeler whether an environmental attribute is represented by an agent or a field. For example, it should be possible to replace a polygon of an area containing a single value representing elevation with a field of the same area with high resolution elevation values. The requirement of the new data model is that it should be able to represent all data that is relevant in environmental models.

What follows is a high level description of our data model. The data model is described in more detail elsewhere (de Bakker et al. [2014]). We describe the *conceptual* data model, which is the data model that the modeler will reason about. The lower level *logical* and *physical* data models may or may not be organized in the same way. Properties of data about the environment that should be representable by the data model are:

- Spatio-temporal location and variation.
- Aggregates with multiple attributes.
- Relations between individuals.
- Uncertainty, both in spatio-temporal location and attribute value.

Figure 1 shows the conceptual data model. At a high level, the spatio-temporal environment can be seen as being occupied by *entities* having *attributes* describing spatio-temporal variation. Examples of such entities are planets, continents, countries, cities, households, trees, sheep, companies, bank accounts, etc. Examples of attributes are name, speed, biomass, size, elevation, etc. Anything that has one or more attributes, or at least a spatio-temporal location is an entity in our data model. Spatio-temporal locations are stored in a *domain*. According to the definition of an entity, relations between

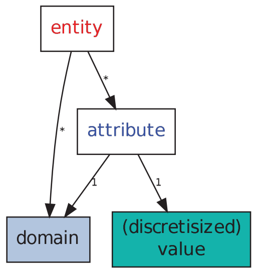

Figure 1. Conceptual data model. Entities refer to spatio-temporal domains and attributes, and attributes refer to domains and (discretisized) values.

entities are entities too. An entity's attribute contains spatio-temporal *values*. Again, spatio-temporal locations of these values are stored in the domain.

Discretization is used to map a continuously varying attribute value to a data structure that can be handled in software. An example of a discretization is the raster, which maps a continuous field to a 2D array of cells containing the field's attribute values. In our data model we consider the discretization as part of the attribute's value.

The resulting data model has four main elements: entity, domain, attribute and value (Figure 1). Each entity has zero or more domains and zero or more attributes. An attribute has a domain and a value. For each spatio-temporal item in the domain the attribute has a value. The domain keeps track of spatio-temporal locations of attribute values. Every time the attribute's value changes, a new item is added to the domain, recording the spatio-temporal location of the new value.

For example, we may want to model a goose entity, with various attributes, like the current location, winter ground, summer ground, migration route and species. The current location may be tracked by GPS trackers attached to the birds, and stored as points in the domain, where each time a new location is measured, a new point is stored in the domain. Attributes using this domain may be tilt, roll and temperature.

This data model is a superset of the traditional raster and vector data models. With this data model is is also possible to model relations between entities, and uncertainty in spatio-temporal location and attribute values. Describing this is outside of the scope of this paper, but more information can be found in de Bakker et al. [2014].

2.2 Syntax

The data model allows us to store spatio-temporal attributes for entities. This corresponds closely to the object data model underlying agent-based modeling environments. One difference is that object behavior is not defined as part of the object. Separating object behavior from its attribute values allows us to design a modeling language that is similar to the language used in field-based modeling environments, and without some of the complexities of object oriented modeling languages used in agent-based modeling environments.

In our language, the majority of a model is made up of statements in which operations are called using a function call syntax (Equation 1a), passing in data about entities and attributes, and returning new values.

$$result_{0,m} = function(argument_{0,n}) \tag{1a}$$
$$selected\_entity\_items = entity[< predicate >] \tag{1b}$$

Using this syntax the modeler can combine built-in generic operations that change the state of entities and their attributes. A number of generic operations can be combined in a user defined function which can be given a name that has a meaning in the context of the model. At the highest level, a model then consists of a list of function call statements, calling user-defined functions that call other user defined functions and/or built-in generic operations. Functions can be combined in modules which can be imported and reused in multiple models.

In agent-based modeling it is common to make selections from the population of agents, based on some criterion. The criterion can be based on spatio-temporal location, or on some predicate involving attribute values. We propose to use the indexing operator for selecting agents based on attribute value predicates (Equation 1b). This is similar to how selection is done in the XML Query language and the netcdf4-python Python module. Selection results in a new entity containing references to domain and attribute information from the original entity, but only for those individuals that are part of the selected set. When selected entity-items are updated, the updates are forwarded to the original entity.

In the next section we will illustrate the syntax shortly described in this section using an implementation of a wolf sheep predation model.

3 Wolf Sheep Predation Case Study

The wolf sheep predation model implemented in this section is based on the model with the same name from the NetLogo models library (Wilensky [1999]). In this model wolves and sheep randomly roam the study area for food. Wolves eat sheep and sheep eat grass. Wandering costs energy, while eating yields energy. Wolves and sheep have a random chance to reproduce. Grass will only regrow after a fixed amount of time. At the start of the model we have to read the initial state of the attributes involved:

```
area = read("area.dat")
sheep = read("sheep.dat")
wolves = read("wolves.dat")
search_radius = 500
sheep_gain_from_food = 15
wolves_gain_from_food = 50
grass_regrowth_time = 14
```

The *area* entity has a *grass* and a *grass_countdown* attribute (Figure 2). These attributes share a domain with bounding boxes. Each of these bounding boxes is associated with a 2-dimensional value discretized as a raster, containing boolean values for the *grass* attribute and integer values for the *grass_countdown* attribute. In the latter case, each cell contains a value representing the amount of time that needs to pass before the cell contains grass again.

The *sheep* and *wolves* entities both have an *energy* attribute (Figure 3). The *sheep.energy* and *wolves.energy* attributes have a domain with point objects. Each of these points is associated with a value representing the energy level of an individual.

Besides the three entities (*area*, *sheep* and *wolves*), the snippet defines four global attributes. Wolves select sheep from an area within the *search_radius*. When wolves and sheep eat, they gain *wolves_gain_from_food* and *sheep_gain_from_food* amounts of energy, respectively. When grass is eaten in a cell, it takes *grass_regrowth_time* amount of time to grow back. Although these four attributes are not attached to an entity, we can think of these global attributes as being part of an anonymous *model* entity. All attributes are part of an entity (Figure 1).

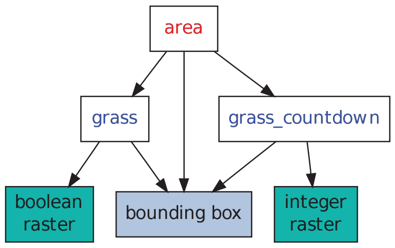

Figure 2. Area entity with two attributes sharing a domain. The colors correspond with the colors used in Figure 1: red: entity, blue: attribute, blue background: domain, green background: value

.

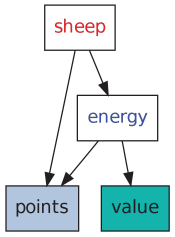

Figure 3. Sheep entity with energy attribute (similar for wolves). Color coding as in Figure 2.

The modeling language allows the modeler to create functions that perform a clearly defined task. A sequence of such function calls is easier to understand than a sequence of the underlying, lower level generic operations. The idea is that the modeler creates higher level functions by combining lower level building blocks.

```
migrate(sheep)
eat_grass(sheep, area, sheep_gain_from_food)
sheep = die(sheep)
reproduce(sheep)

migrate(wolves)
eat_sheep(wolves, sheep, search_radius, wolves_gain_from_food)
wolves = die(wolves)
reproduce(wolves)

grow_grass(area, grass_regrowth_time)
```

This section of the model shows some of the individual steps that are performed for each time step. For both wolves and sheep the steps involve migrating, eating, dying and reproducing. After that the grass attribute is updated. The high-level functions that implement the model are all short aggregates of lower level operations:

```
def migrate(animals):
    move(animals, ...)
    animals.energy -= 1
```

In this function, each individual animal is moved and each animal's energy level is updated. *move* is a built-in operation that updates the spatial coordinates of the entity passed in. Since the entity passed in has only one domain (Figure 3), it is used by the *move* operation. In other cases an attribute or a domain must be passed in. The entity referred to by *animals* is updated.

```
def eat_grass(sheep, area, gain_from_food):
    sheep_on_grass = sheep[area.grass]
    sheep_on_grass.energy += gain_from_food
    area[sheep_on_grass].grass = False
    area[sheep_on_grass].grass_countdown = grass_countdown
```

The selection operator ([]) is used to select a subset of sheep, based on a predicate using an external (to the sheep entity) attribute. A new entity is created that contains all attributes of the original entity. For each attribute, a selection is made of those domain objects that overlap with the *area.grass* attribute's domain for which the attribute's value evaluates to True. *area.grass* has a 2-dimensional attribute value, and only those sheep are selected that are located in cells containing a True value. *grass_countdown* is assigned to those locations in the *area.grass_countdown* attribute that correspond with the locations of sheep objects. The *sheep* and *area* entities passed in are updated.

```
def eat_sheep(wolves, sheep, search_radius, gain_from_food):
    wolves_near_sheep = select_by_radius(wolves, sheep,
        search_radius)
    wolves_catching_sheep = select_random(wolves_near_sheep, ...)
    nr_sheep_eaten_per_wolf = 1
    caught_sheep = select_by_radius(sheep, wolves_catching_sheep,
        search_radius, nr_sheep_eaten_per_wolf)
    delete(caught_sheep)
    wolves_catching_sheep.energy += gain_from_food
```

Wolves are selected by using a search radius around each wolf. A random selection of these wolves eat one of the nearby sheep. Multiple wolves can share the same sheep.

4 Conclusion and future work

In this paper we have presented the first steps towards a simple, imperative modeling language for defining models in which both agents and fields are manipulated. We described a new data model, in which the traditional difference between vector and raster data is not that pronounced anymore. This leads to a uniform treatment of environmental data by the modeling language. The data model organizes data in a similar fashion as agent based models do for agents. In our case the behavior of the agents is modeled separately from the data, though. This has the benefit of being able to design a simple modeling language that is comparable to the current modeling languages used in field-based modeling, which was the goal of this study.

Due to space constraints, we have skipped over many aspects in this paper. We did not describe how iteration through time can be handled, how events can be handled, how to go about positional and value uncertainty, and how to model relations, for example. We do recognize the importance of these aspects, though. Also, although we presented an example model in this paper, we do not yet have the software to execute such a model. We are currently working on a prototype implementation of the modeling language.

5 Acknowledgements

We would like to thank two anonymous referees for reviewing an earlier version of this paper and providing us with valuable suggestions for improving it.

References

Crooks, A. T. and Castle, C. J. (2006). Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations. Technical Report 0, UCL Centre for Advanced Spatial Analysis, London.

de Bakker, M. P., de Jong, K., and Karssenberg, D. (2014). A Conceptual Data Model for Integrating Fields and Agents. In Ames, D. P. and Quinn, N., editors, *Bold Visions for Environmental Modelling, Seventh Biennial Meeting, San Diego, California, USA*, International Congress on Environmental Modelling and Software. International Environmental Modelling and Software Society (iEMSs).

Eastman, J. R. (2009). *IDRISI Guide to GIS and Image Processing*. Clark University, Worcester, MA, selva edition.

Filatova, T., Verburg, P. H., Parker, D. C., and Stannard, C. A. (2013). Spatial agent-based models for socio-ecological systems: Challenges and prospects. *Environmental Modelling & Software*, 45:1–7.

Goodchild, M. F., Yuan, M., and Cova, T. J. (2007). Towards a general theory of geographic representation in GIS. *International Journal of Geographical Information Science*, 21(3):239–260.

Karssenberg, D., Schmitz, O., and de Jong, K. (2012). Stochastic spatio-temporal modelling with PCRaster Python. In Abbasi, A. and Giesen, N., editors, *EGU General Assembly Conference Abstracts*, volume 14 of *EGU General Assembly Conference Abstracts*, page 4367.

Macal, C. M. and North, M. J. (2009). Agent-based modeling and simulation. *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 86–98.

Neteler, M., Bowman, M. H., Landa, M., and Metz, M. (2012). GRASS GIS: a multi-purpose Open Source GIS. *Environmental Modelling & Software*, 31:124–130.

Wesseling, C. G., Karssenberg, D., Burrough, P., and van Deursen, W. P. A. (1996). Integrating dynamic environmental models in GIS: The development of a Dynamic Modelling language. *Transactions in GIS*, 1(1):40–48.

Wikipedia (2014). Comparison of agent-based modeling software - Wikipedia, the free encyclopedia. http://een.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software. Accessed March 3, 2014.

Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo.

Zandbergen, P. A. (2013). *Python Scripting for ArcGIS*. Esri Press.