Ecological Modelling xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Ecological Modelling

journal homepage: www.elsevier.com/locate/ecolmodel

Design of a Domain Specific Language for modelling processes in landscapes

P. Degenne[a,*], D. Lo Seen[a], D. Parigot[b], R. Forax[c], A. Tran[a], A. Ait Lahcen[b], O. Curé[c], R. Jeansoulin[c]

[a] CIRAD, UMR TETIS, Montpellier F-34398, France
[b] INRIA, Sophia Antipolis F-06902, France
[c] Institut Gaspard Monge, Université Marne-la-Vallée F-77454, France

ARTICLE INFO

Article history:
Available online xxx

Keywords:
Ocelet
DSL
Multi-scale
Spatial and temporal modelling
SOA

ABSTRACT

The modelling of processes that occur in landscapes is often confronted to issues related to the representation of space and the difficulty of properly handling time and multiple scales. In order to investigate these issues, a flexible modelling environment is required. We propose to develop such a tool based on a Domain Specific Language (DSL) that capitalises on the service-oriented architecture (SOA) paradigm. The modelling framework around the DSL is composed of a model building environment, a code generator and compiler, and a program execution platform. The DSL introduces five language elements (*entity*, *service*, *relation*, *scenario* and *datafacer*) that can be combined to offer a wide range of possibilities for modelling in space and time at different scales. When developing a model, model parts are either built using the DSL or taken from libraries of previously built ones, and adapted to the specific model. The practical usage of the DSL is illustrated first with the Lotka–Volterra model, and then with a landscape modelling experiment on the spread of a mosquito-borne disease in the Sahelian region of West Africa. An interesting characteristic of this approach is the possibility of adding new elements into an existing model, and replacing others with more appropriate ones, thus allowing potentially complex models to be built from simpler parts.

1. Introduction

Computer modelling of systems in space and time is common practice in many scientific disciplines. It allows by simulation the verification of the knowledge one has of a system, and therefore helps to better understand how the system works in some situations, while aiming at predicting the behaviour of the system in a variety of other situations. When the system considered is a landscape, for which full scale physical experimentation can rarely be considered, modelling could be applied to help analyse a variety of important issues facing society today, such as the degradation of natural ecosystems with loss of biodiversity, the emergence and spread of new diseases due to changing environmental and climatic conditions, or the uncontrolled urbanization and population migrations as expressions of deep social transformations.

The modelling of spatial and non-spatial dynamics of landscapes have been carried out in a large variety of not only thematic, but also methodological contexts. No less than five paradigms or modelling formalisms – system dynamics (SD), discrete event (DE), cellular automata (CA), agent-based (AB) and geographic information systems (GIS) – are being used (Burrough and McDonnell, 1998; Borshchev and Filippov, 2004; Bousquet and Le Page, 2004; Ratzé et al., 2007). This diversity, while being a sign of an active research field, may also suggest that the concepts used by modellers could be too diverse to be satisfactorily described with any single formalism. For instance, while geared for manipulating spatial information, GIS suffer from an intrinsic limitation of not properly handling time (e.g., Langran, 1992). During the last two decades there have been major contributions to address the Time issue in GIS (e.g., Langran, 1992; Peuquet, 1994; Worboys, 1994; Claramunt and Thériault, 1995; Yuan, 1999; Wachowicz, 1999; Parent et al., 2006). Adding Time as another dimension to space proved however not to be just an implementation problem, and recommendations were made that more theoretical and conceptual developments would be required (Peuquet, 2001). Likewise, formalisms that consider Time first (i.e. SD, DE) face the opposite limitation with spatial information, where it is widely assumed the latter can only be treated as either field or object models (Goodchild, 1992; Peuquet, 2001). Improvements were sought with coupled or hybrid models that capitalise on more than one of the formalisms: SD-DE (e.g., Ziegler et al., 2000); AB-SD (e.g., Duboz et al., 2003); GIS-AB (e.g., Brown et al., 2005; Torrens and Benenson, 2005); AB-DE (e.g., Uhrmacher and Schattenberg, 1998); AB-CA (e.g., Bousquet et al., 1998); CA-DE (e.g., Wainer and Giambiasi, 2001). These works are representative of what can be considered a highly active research domain, where research communities assemble to address common thematic (e.g. landscape ecology, urban plan-

* Corresponding author.Tel.: +33 467548713; fax: +33 467548700.
E-mail address: pascal.degenne@cirad.fr (P. Degenne).

0304-3800/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ecolmodel.2009.06.018

ning and management, spatial epidemiology), methodological (e.g. MAS—multi-agent systems, DEVS—discrete event system specification) as well as conceptual (e.g. object-field models of space (e.g., Couclelis, 1993; Cova and Goodchild, 2002), hierarchy and scales (e.g., Wu, 1999), data quality (e.g., Devillers and Jeansoulin, 2006), indeterminate boundaries (e.g., Burrough and Frank, 1996), time in GIS) issues.

In addition to the problem of choosing the appropriate modelling approach in a given context, previous studies have stressed on the difficulties that modellers face when working from conceptual models of dynamic landscapes to their simulation on a computer (Fall and Fall, 2001). A general-purpose modelling language such as UML (OMG, 2009) appears unsuitable for two main reasons: (i) it is not a directly executable specification: the execution model is only partially implemented, such that the user must manually complete the produced code and (ii) the concepts proposed are very general, and not readily configurable to the present case. One approach has been to develop domain specific languages (e.g. SME; Maxwell and Costanza, 1997) (SELES; Fall and Fall, 2001) (L1; Gaucherel et al., 2006) that would allow domain experts to concentrate on the conceptual model, while leaving to an associated software tool the transformation of the model into an implementation that runs on a computer. In this way, domain experts may develop models using a higher level language, instead of programming directly with general-purpose languages like Java or C++. However, for such a large domain where spatial, temporal and multi-scale issues are still actively being studied, a DSL that can support research on modelling processes in landscapes has to be flexible, and especially so at the very basic level where landscape features and their interactions are defined. For example, a DSL that has originally been developed using a predefined spatial data structure (e.g. grid cells) may limit modellers in situations where other structures are more appropriate (Gaucherel et al., 2006). A trade-off between ease of use and expressiveness of a DSL therefore seems inevitable here.

An interesting parallel can be made between, on one hand, landscape entities and their interactions that need to be modelled, and on the other, software components and services that emerged with the *c* omponent model programming paradigm in the nineties (Szyperski, 1998). The latter was developed to help design and maintain increasingly complex software applications, where the object-oriented approach was starting to show its limits (Marvie and Pellegrini, 2002; Courbis et al., 2004). With the development of Internet (e.g. web services W3C, 2002), applications had to be *distributed* (as opposed to centralised), and able to interact with many other applications *dynamically* (thus forming a graph of relations that changes with time). Communicating software components in many aspects behave like interacting features in a landscape, and it is not surprising that many notions used when modelling processes occurring in landscapes, such as dynamics, delays, events, fluctuations, response or agent behaviour, are also present in the service-oriented architecture paradigm (SOA; Papazoglou et al., 2007).

In this study, we present a DSL that is being developed for experimenting the modelling of a variety of landscape situations. Compared to similar existing languages, we have defined language elements that seek a balance between modelling facility in simpler situations and adequate expressiveness in more complex ones, while taking advantage of the flexibility offered by component-service programming. The structure and the logic of the language, as well as the language elements, are first introduced. The Lotka–Volterra predator–prey model is then used to illustrate the different steps between a model written in the DSL and a running simulation. Finally, a possible way of using the DSL to model a more complex situation is presented, through a landscape modelling experiment on the spread of a mosquito-borne disease in an arid area in West Africa where ponds, pastures, herds and mosquitoes come into play. The focus will be less on the model details than on the way the main principles underlying this approach are expressed.

2. The Ocelet modelling language and its main concepts

Ocelet has followed DSL development procedures recommended by Mernik et al. (2005). Its design had to meet two main requirements: (i) it has to provide concepts adapted for modelling processes in landscapes and (ii) it must have underlying operational semantics that are able to automatically generate code and run simulations corresponding to the models written with the language. Around the language there is a modelling framework composed of

- a model building environment that enables syntax analysing and type verification,
- a code generator and compiler,
- a program execution platform based on component-service technologies (Fig. 1).

Ocelet is designed around five main concepts: *entity, service, relation, scenario* and *datafacer*. We define hereafter how these concepts should be understood in the context of Ocelet. Other common concepts such as argument, property, number are also used, but they do not require specific descriptions.

2.1. Entity

Entities are basic modelling parts that can be put together to build a model. A whole model is, as such, also an entity. An entity can contain other entities, and is then called a *composite* entity. Entities that do not contain other entities are called *atomic* entities. A forest for example can be modelled by a composite entity that contains tree entities which are part of the forest.

From a computer science point of view, an entity is a component: an independent piece of code that can be connected to other components to build an application. Entities can perform operations called services. Entities being software components, they can dynamically be connected through their services, even without knowing how they are designed internally.

Structure of an entity:

```
entity(name, property*, service*, entity*, sce-
nario*, relation*, datafacer*)
```

That specification means that an entity can contain properties (`property*` means 0 or more property), services, entities, scenarios, relations, datafacers, and a name.

2.2. Service

A service is a functional description of how one can relate to an entity. It is thus a communication port of an entity. As arguments, service accept values from other entities, and describes the capability of the entity to export values to other entities of the model. Services are published outside the entity they belong to, meaning that it is possible to obtain a list of all the services an entity provides.

Structure of a service:

```
service(name, argument*,result)
```

Services are defined like functions: they have a name, accept arguments, can produce results.

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

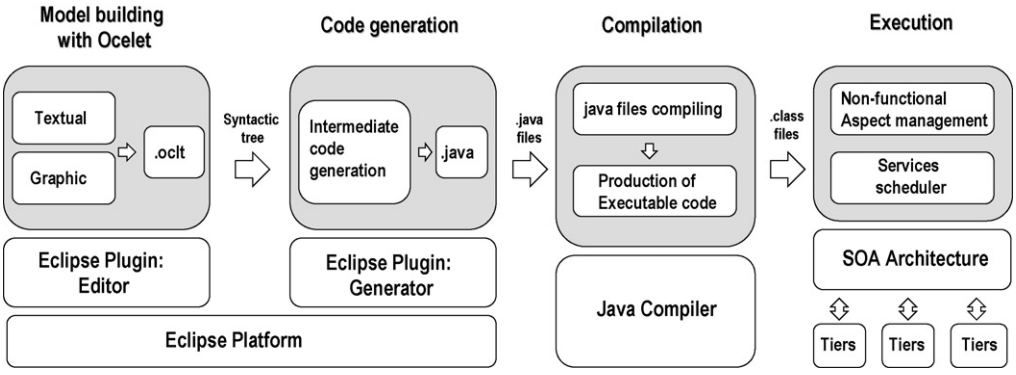

Fig. 1. The *Ocelet* modelling and simulation framework.

2.3. Relation

An entity can directly call a service available on another entity. It is the simplest link that can be established between two entities. When modelling interacting landscape elements, relations often cannot be reduced to just a transfer of information between entities. Information sometimes needs to be transformed according to the nature of the relation. Two aspects of the interactions have to be considered: we have to indicate which entities are interacting with each other, as expressed by an interaction graph, and at the same time describe what is happening when they interact. In Ocelet, the concept of Relation integrates both of these aspects.

A relation will apply to a *selection* of entities based on compatibility of services. From that selection, only those fulfilling certain conditions will be subject to a number of *actions* that may affect other entities. Selections and conditions can be defined according to spatial as well as temporal criteria. Actions can typically be the creation, destruction or modification of an entity or a relation, or the compositing or degrouping of entities. Functional and spatial relations are often needed when modelling landscapes. For example, grazing can be seen as a functional relation between herds and pastures, linking "surface area" of pastures entities to "biomass ingested" of herds entities. Spatial relations can be used to give a topological description of a landscape, like for example, field A has common boundaries with fields B and C.

Structure of a relation:

```
relation(name, interface, interface, action*)
interface(name, service*)
action(operation*)
```

In the definition above, *interface* is referred to as a set of services, and "operation" is an elementary part of a process describing the exchanges taking place when the entities interact with each other. Operations can be service calls, but also arithmetical instructions or tests. An *action* is carried out using several operations.

Example definition of a "grazing" relation:

```
relationdef Grazing(grass, herbivores)
{
number value=grass.getSurfaceArea();
herbivores.addBiomassIngested(value*conversioncoeff);
}
```

Any entities having "grass" and "herbivores" interfaces will be able to accept "grazing" relations. In this example, the relation uses grazed "surface area" information from an entity having a "grass" interface, and convert it (here, simply using a conversion coefficient) to an "ingested biomass" information that is sent to another entity having a "herbivores" interface.

2.4. Scenario

A scenario gives a description of which actions and relations within a composite entity have to be activated, and when. The relations in turn put selected entities in interaction in space and time. The scenario therefore expresses the spatial and temporal internal behaviour of a composite entity by managing the entities and relations it contains. For example, a ten-year evolution scenario embedded in a village entity could describe the extension of the village by a few houses every year, taking in account population growth and several policy rules that govern spatial expansion. The ten-year scenario could also be composed of yearly evolution scenarios.

Structure of a scenario:

```
scenario(name, operation*, scenario*)
```

In practice, a scenario can be used to describe how an entity evolves undisturbed for a given time period, and another scenario can contain the behavior of the same entity when a disturbance event arises.

2.5. Datafacer

A datafacer is a device through which entities access data. The data can be in the form of an external database or satellite image, but can also be internally generated, like in a logfile, during model execution. The datafacer contains the necessary functions, developed for specific types of data sources, to provide the services required by the entity to which it is attached. The other entities of the model can interact with the Datafacer in a coherent manner whithout having to deal with the details of how data access and queries are made. More formally, a Datafacer is an atomic entity that can be accessed directly by any entity in a model.

2.6. How these concepts work together

Modellers who understand the landscape "system" they study as interacting landscape elements, should be able to express their understanding with Ocelet without much compromise. Landscape elements are modelled as entities, which in turn can contain other landscape elements (entities). Interactions between landscape elements are modelled using relations. The latter are not just "wires" for transferring information, but can also hold instructions on what to do when entities are in relation, thus expressing the "nature" of the relations.

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

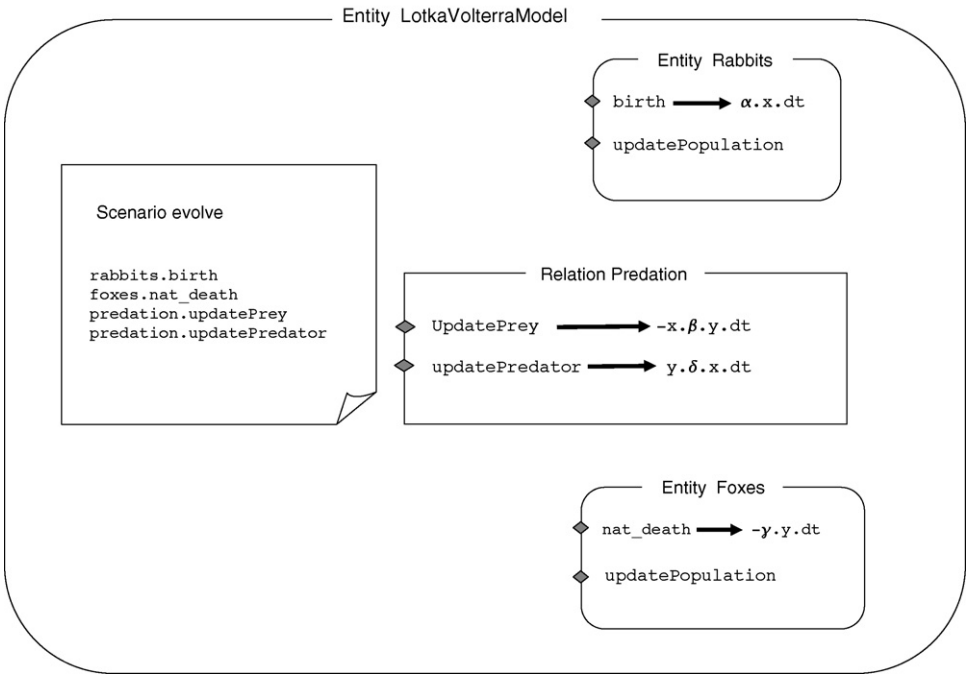

Fig. 2. Lotka–Volterra as modelled with two entities (Foxes, Rabbits) and one relation (Predation).

The orchestration of the timing of the interactions between elements in a landscape (modelled as entities contained in an entity) is carried out in a scenario attached to the landscape (container entity). The services of an entity express the behaviour of that entity as seen from outside. Datafacers are a convenient way for entities in a model to access heterogeneous data sources through a unique mechanism based on services, and in coherence with the rest of the language.

3. Two examples to illustrate the Ocelet approach

The Ocelet language elements have been designed to be generic enough to express diverse concepts that are needed when modelling processes in landscapes. To draw an exhaustive list of all the possible ways of combining these language elements is, however, not realistic. Instead, we have chosen to illustrate with two examples how to use the language, and to highlight some possibilities it offers. One example is the well known Lotka–Volterra model, which is used here to show a first application of Ocelet, from model building to running simulations. The other example involves a mosquito-borne disease. The modelling of the processes related to the disease is quite challenging, requiring appropriate field data collection (meteorological, hydrological, socio-economic, entomological, serologic, ...), the development, validation and testing of different models (hydrological, epidemiological, ...), and is therefore not within the scope of the present paper. But the context it draws is relevant for showing a few interesting characteristics of the Ocelet approach: modelling a system with functional relations, the use of a datafacer to access spatial data, and the possibility of incrementally enriching a model.

3.1. Implementing the Lotka–Volterra model

The Lotka–Volterra prey–predator model (in Murray, 2003) aims at calculating the evolution of the size of two animal populations in interaction, one prey (noted x in the equations) and the other predator (y). The model is a couple of first order differential equations:

$$\begin{cases} \dfrac{dx}{dt} = x.(\alpha - \beta y) \\ \dfrac{dy}{dt} = y.(-\gamma + \delta x) \end{cases} \quad (1)$$

where α is an expression of the birth rate in the prey population, β is the death rate of prey due to predation, γ represents the natural death rate in the population of predators, δ is the rate of predator population growth per prey consumed.

With Ocelet, we can model Lotka–Volterra using two entities (Foxes for predators and Rabbits for prey) and one relation (Predation). Each of these entities has an internal behavior describing the natural evolution of the population under normal conditions: birth of rabbits, and death of foxes. The `Predation` relation models how both populations are affected when the prey and predators meet. A connection operator: `Predation.connect(foxes, rabbits)` establishes a functionnal link between foxes and rabbits through the `Predation` relation (see Appendix A for a sample Lotka–Volterra model written in Ocelet). The time flow of the system is described in the `evolve` Scenario.

As shown in Fig. 2, the differential equations of system (1) are split in two parts. The $x.\alpha$ and $-y.\gamma$ are respectively calculated by the `birth()` service of Rabbits entity, and the `natural_death()` service of Foxes entity. The $-x.\beta.y$ and $y.\delta.x$ terms are calculated only for the entities that have been connected to each other through the `Predation` relation. Note that their expressions are stored inside the relation itself, which is one originality of Ocelet. The idea behind this is to be able to reuse already developed relations with other entities, and in the same line, to consolidate a library of ready made relations (or entities, scenarios and datafacers) to facilitate future model development. A prototype compiler was developed for the Ocelet textual syntax, and an execution environment (or runtime) was created. The Lotka–Volterra model written in Ocelet was thus translated into Java. The resulting Java classes obtained were run using the execution environment to produce the results shown in Fig. 3.

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

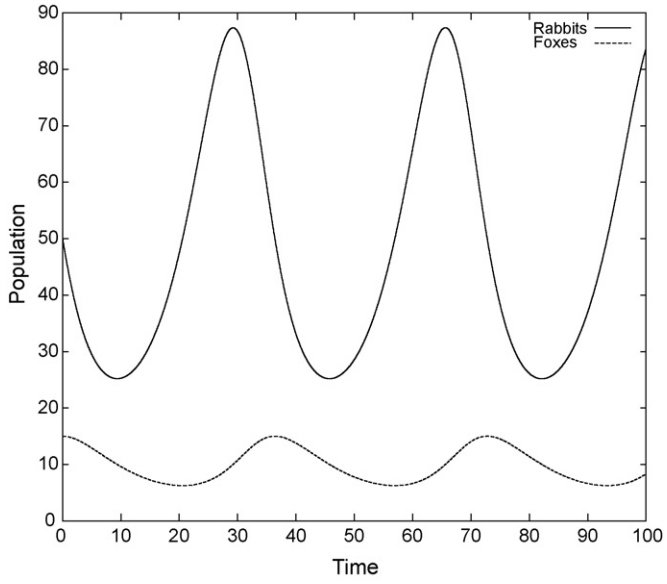

Fig. 3. A simulation of the Lotka–Volterra model developed with Ocelet ($\alpha = 0.1; \beta = 0.01; \gamma = 0.05; \delta = 0.001$).

3.2. Experimenting with Ocelet on a mosquito-borne disease

The spatial and temporal distribution of mosquitoes responsible for various vector-borne diseases are often linked to landscape dynamics, as mosquitoes require appropriate breeding sites for their development. One important such disease is the Rift Valley Fever (RVF) which affects both livestock and humans. In livestock, outbreaks are generally associated with mass abortions and high mortality rates in young animals, and may result in important economic losses. The transmission of the virus in the Sahelian region of North Senegal is related to the dynamics of temporary ponds which are favorable mosquito larval habitats. The livestock production system of the region is extensive and during the rainy season, areas in the vicinity of permanent or temporary ponds are used by transhumant herds for water and grazing needs (Bah et al., 2006). When trying to model the spread of the virus, present models, mainly epidemiological, solve ordinary differential equations (ODE) for different populations of mosquito species (Gaff et al., 2007). Most of the spatial nature of the complex problem is, however, either ignored, or concealed in appropriate contact rate parameters that are difficult to estimate. As far as we know, only few studies focused on the spatial dynamics of vectors and the disease they may transmit (Tran and Raffy, 2006; Otero et al., 2008; Linard et al., 2009). In order to understand the dynamics of the disease in view of proposing control measures, any important aspect of the problem must not be ignored: it would be necessary to model mosquito populations according to pond dynamics and presence of livestock, and therefore also model ponds, pastures, herds that move following availability of water and food, and the transmission of the virus to the animals. The approach that we are exploring offers interesting possibilities for modelling a complex problem by focusing on each part one by one, without ignoring the interactions between the parts. In the next sub-sections we focus on some of these possibilities.

3.2.1. Modelling simple pond dynamics

In and around a given pond, the presence and abundance of *Aedes* and *Culex* mosquitoes at different life stages depend for a large part on the sequence and duration of wet and dry periods for that pond. Here we start with a simplified model of pond dynamics that describes the evolution of water surface, given the pond's shape and the quantity of water incoming or leaving the pond. The positive and negative terms of a pond's water budget are assumed to be only rainfall and evaporation respectively. Other terms such as infiltration, run-off or water consumed by animals have been ignored in this example, but could be included in a similar way. Therefore to start with, the model is made of three kinds of entities: two atomic (Pond and Meteo) and one composite (the model itself). The functioning of the model will rely on the relations between these entities, and on the scenario that describes how these relations are expressed in time.

As shown in Fig. 4, the Pond entity only needs two services: `waterIncome()` and `evaporate()`. The first uses rainfall to calculate the amount of water the pond receives, the second takes other meteorological variables (solar radiation, wind speed, etc...) to estimate evaporation. These data would be obtained from a Meteo entity which also provides two services: `rainfall()` and `otherMeteoVar()`. A `waterExchange` relation models how one pond entity updates its water budget given the meteorological data obtained from a meteo entity. The relation is described as a one to one interaction between a meteo and a pond. But when initializing a simulation, it is likely that many instances $pond_i$ of the Pond entity will be created, each with different shapes and locations. Typically, a series of calls to a `waterExchange.connect(meteo,pond`$_i$`)` is needed to establish a link between one instance of Meteo entity and every $pond_i$ entity through the `waterExchange` relation. The `evolve` scenario will be executed for every time step of the simulation. That scenario is based on a `select` statement (see Fig. 4) that will apply the `updatePond()` service of the relation to all the entities that had previously been connected. In other words, the series of calls to a `connect()` statement creates an *interaction graph* between one meteo entity and many $pond_i$ entities, and once that graph is built, a call to `updatePond` on the relation is enough to update all the ponds present in that interaction graph. The purpose of the `select` statement is to provide a way to activate only a subset of the interaction graph. For example, one can imagine a selection based on spatial attributes that would call `updatePond()` on all the ponds located in a given area.

3.2.2. Datafacer to locate ponds

When creating many instances of the Pond entity while building the pond dynamics model, the specific shape and location of each pond can be obtained from an existing GIS file (a shapefile for example). The initialising scenario of the model needs to access the source of data to obtain the unique parameters of every Pond it creates. The Datafacer is an entity specialised in the access of external data sources and offer services that can be called by entities within the model. For the present example, we can use a `shapefileDatafacer`. When added to the model, the `shapefileDatafacer` is given a few parameter settings, like the name and location of the shapefile, and some metadata needed to access the right attributes from the file.

The datafacer is then used in the model like any other entity. It can be linked to other entities through relations, or its services can be accessed directly. Moreover, it can sometimes be convenient to use a datafacer for the management of space when the type of spatial operations needed is very efficiently implemented by the data management system it is associated with. A call to a service of the `shapefileDatafacer` could be one way to make a selection based on spatial attributes for example.

3.2.3. Extending an existing model

The simple pond dynamics model presented earlier can be made more realistic by improving the description of its parts (which can be of different types: entity, relation, datafacer or scenario) without changing the logic of the model. When studying the RVF problem, however, more processes are also to be considered, among which,

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

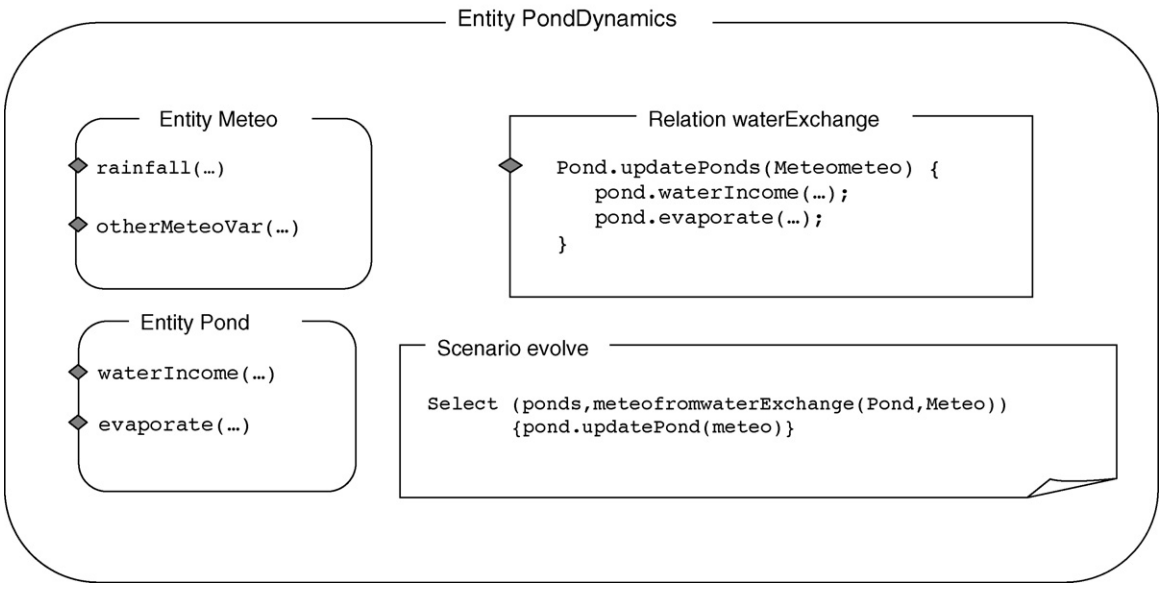

Fig. 4. Main elements of a simple pond dynamics model.

the dynamics of pastures, the displacement of herds between ponds and grazing areas, the development of mosquito populations in the ponds, and the transmission of the RVF virus. From the point of view of livestock management, it may be enough to know how the grazing areas and ponds are changing during the season. As disease surveillance by veterinary services are mainly based on farmer reports, farmers can only estimate an a posteriori risk of animals being infected near the ponds. The point of view of the entomologist, with a good understanding of mosquito population dynamics in the ponds, and that of the epidemiologist, with the knowledge of how the virus is transmitted, would be needed to better estimate this risk.

The inclusion of a mosquito model within the pond models can be done through an *incremental modelling process*. Once the simple pond model has been tested and considered satisfactory, the atomic pond entities can be replaced by composite pond entities that contain mosquito population entities at different stages of their life cycle (egg, larva, pupa and imago). *Aedes* and *Culex* mosquitoes are modelled separately, with different behaviours with respect to disease transmission (respectively initiators and amplifiers), and also to climatic conditions. That second version of pond entities would provide at least the same services as the first version to make sure that the new more complex ponds would seamlessly integrate and interact with the rest of the landscape model. This incrementation process can be carried out further, as other services and relations need to be included so as to take into account the interaction between mosquitoes and animals. Mosquito population entities also need to contain viral capacity entities from which the transmission cycle of the disease (incubation time, virulence of the virus, etc.) would stem.

4. Discussion

When using this approach, modellers in the beginning may find that the conceptual leap between the elements of the language and usable modelling parts is too large, although the initial objective was to model using concepts closer to the thematic domain. This seems inevitable, given the large variety of concepts that are needed in different thematic areas, and this problem is also common to present modelling approaches. However, we rely on the "reusability" of the modelling primitives to ease this problem. Here, the term "primitive" refers to a piece of code in Ocelet, containing an entity, a relation, a scenario or a datafacer, that had previously been written for a given case, and that could be used again for another. When numerous different situations will have been modelled with this approach, we expect that a set of most useful types of primitives will emerge, from which modellers would pick and adapt to their case studies. This also implies that primitives building would always be part of the modelling exercise, although most users would rarely need to do it completely from scratch except for very specific cases.

It may be argued that as more and more elements are added in a model, at some point, computing efficiency may become an issue. It is probably too early to have a clear stand on this, but there are at least two points on which efforts can be focused. First, computing efficiency will depend on the quality of the code generated, and therefore on the code generator itself. As required, code generators can be developed towards general purpose languages (C++, or Java as the one used in Section 3.1) and also be optimised for specific platforms (parallel, distributed). Part of the computing efficiency issue can be addressed there. The larger part, however, should be handled with good model design. This approach offers quite a large freedom of expression in modelling, and the possibility of improperly combining the primitives is not to be neglected. Acquired experience and an evolving set of best practices may help to avoid this pitfall.

On where this approach stands with respect to other paradigms used in modelling landscapes, the present one, built on a language and a simulation platform, is meant to be a tool for investigating present limitations when expressing the behaviour of landscapes in space, time and at different scales. It does so not quite by proposing direct solutions for each and every limitation encountered in the paradigms (which in any case would not be very realistic) but rather with an alternative way of hierarchising problems, by separately containing them, without breaking the links of their interconnections. Each problem could then be dealt with appropriately, by using if necessary the solutions present in existing paradigms. In the example above, we have seen datafacers that communicate with GIS, entities that behave like in system dynamics, and scenarios that can be formulated as discrete events. We could also add that the herds, through the adaptive decisions of the herdsmen, clearly need to exhibit agent-like behaviour. This approach could

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

then be used to investigate heterogeneous systems and could, as such, be considered a versatile complement to existing modelling approaches.

5. Conclusion

This paper introduces the Ocelet modelling language, together with a description of the main concepts on which it is based. Five language elements (entity, service, relation, scenario and datafacer) can be combined to offer a wide range of possibilities for modelling in space and time at different scales. In this modelling approach, primitives are either built using the DSL, or taken from libraries of previously built primitives. For instance, when starting on a new model, the modeller has access to a number of most frequently used primitives that can be adapted for the specific model. One important characteristic of this approach is the flexibility of adding new elements into an existing model, and of replacing elements with more appropriate ones. In this way, potentially complex situations can be modelled by assembling simpler parts.

The ongoing phases in the DSL development are its implementation and deployment. The main step in the implementation phase is the development of a code generator, which is a translator of models written in the DSL into a general purpose language (here, Java). Another important step is the development of the GUI (Graphical User Interface) to help in model building, in an Eclipse-like environment (Eclipse, 2009). But the most important factor which will determine user adoption of this approach, is how rich the primitives libraries are. This is where our development efforts are currently being focused when working on modelling a number of very different situations. Although more work is required for the development of the modelling framework to be complete, we believe that the approach presented already suggests stimulating prospects on the way landscapes can be modelled. In the end, more than the specific language that we are proposing, it is probably the new way of embracing a modelling problem that could constitute a fair contribution of this paper.

Acknowledgements

This work was supported (in part) by the "Blanc" Program of the *A* gence Nationale de la Recherche (ANR) under Project No. ANR-07-BLAN-0121 (STAMP: modelling dynamic landscapes with spatial, temporal and multi-scale primitives). The main ideas presented in this paper have been discussed during several working sessions. The contributions of D. Auclair, C. Baron, C. Gaucherel, C. Proisy, G. Roussel and V. Soti are gratefully acknowledged. An oral presentation of this work was made by PD at the "Spatial Landscape Modelling" Symposium, Toulouse (France), in June 2008. We thank the anonymous reviewers who have helped us improve a previous version of the manuscript.

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

Appendix A. Lotka–Volterra model written in Ocelet (a first version of its textual concrete syntax)

```
entity LotkaVolterraModel {

   configuration {
     number alpha = 0.1;
     number beta = 0.01;
     number gamma = 0.05;
     number delta = 0.001;
     number NbR0 = 50;
     number NbF0 = 15;
     number dt = 0.0025;
   }

   entity Rabbits {
     property number nbPopulation;

     service birth() { nbPopulation = alpha*nbPopulation*dt; }

     service updatePopulation(number difference) {
       nbPopulation = nbPopulation + difference;
     }
   }

   entity Foxes {
     property number nbPopulation;

     service natural_death() {
       nbPopulation = - gamma * nbPopulation * dt;
     }

     service updatePopulation(number difference) {
      nbPopulation = nbPopulation + difference;
     }
   }

   relationdef Predation(Predator, Prey) {
     assume property number Predator.nbPopulation;
     assume property number Prey.nbPopulation;
     assume service Predator.updatePopulation(number difference);
     assume service Prey.updatePopulation(number difference);

     service Predator.updatePredator(Prey prey) {
       updatePopulation(nbPopulation + delta * old nbPopulation *
       prey.nbPopulation * dt);
     }
     service Prey.updatePrey(Predator predator) {
       updatePopulation(nbPopulation - beta* old nbPopulation *
       predator.nbPopulation * dt);
     }
   }

   universe {
     let global rabbits = Rabbits { nbPopulation = NbR0; };
     let global foxes = Foxes { nbPopulation = NbF0;};
     relation Predation(Foxes, Rabbits);
       Predation.connect(foxes, rabbits);
   }

   scenario evolve {
     rabbits.birth();
     foxes.natural_death();
     select (foxes, rabbits from Predation(Foxes, Rabbits)) {
       foxes.updatePredator(rabbits);
       rabbits.updatePrey(foxes);
     }
   }
}
```

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018

References

Bah, A., Touré, I., Le Page, C., Ickowicz, A., Diop, A.T., 2006. An agent-based model to understand the multiple uses of land and resources around drillings in Sahel. Mathematical and Computer Modelling 44, 513–534.
Borshchev, A., Filippov, A., 2004. From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools. In: Proceedings of the 22nd International Conference of the System Dynamics Society, July 25–29, Oxford, England.
Bousquet, F., Bakam, I., Proton, H., Le Page, C., 1998. Cormas: common-pool resources and multi-agent systems. In: Pobil, A.P.D., Mira, J., Moonis, A. (Eds.), Lecture Notes in Artificial Intelligence, vol. 1416, pp. 826–838.
Bousquet, F., Le Page, C., 2004. Multi-agent simulations and ecosystem management: a review. Ecological Modelling 176, 313–332.
Brown, D.G., Riolo, R., Robinson, D.T., North, M., Rand, W., 2005. Spatial process and data models: toward integration of agent-based models and GIS. Journal of Geographical Systems 7, 25–47.
Burrough, P.A., Frank, A.U. (Eds.), 1996. Geographic Objects With Indeterminate Boundaries. Taylor & Francis, London, 345 pp.
Burrough, P.A., McDonnell, R.A., 1998. Principles of Geographical Information Systems. Oxford University Press, Oxford.
Claramunt, C., Thériault, M., 1995. Managing time in GIS: an event-oriented approach. In: Clifford, J., Tuzhilin, A. (Eds.), Recent Advances on Temporal Databases. Springer-Verlag, Zurich, pp. 23–42.
Couclelis, H., 1993. People manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS. In: Frank, A.U., Campari, I., Formentini, U. (Eds.), International Conference GIS—From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning. Springer-Verlag, Berlin, pp. 65–77.
Courbis, C., Degenne, P., Fau, A., Parigot, A., 2004. Un modèle abstrait de composants adaptables. revue TSI. Composants et adaptabilité 23 (2), 231–252.
Cova, T.J., Goodchild, M.F., 2002. Extending geographical representation to include fields of spatial objects. International Journal of Geographical Information Science 16 (6), 509–532.
Devillers, R., Jeansoulin, R. (Eds.), 2006. Fundamentals of Spatial Data Quality, France. ISTE, London, ISBN 1905209568, 320 pp.
Duboz, R., Ramat, E., Preux, P., 2003. Scale transfer modeling: using emergent computation for coupling an ordinary differential equation system with a reactive agent model. Systems Analysis Modelling Simulation 43, 793–814.
Eclipse, 2009. Eclipse Development Framework: http://www.eclipse.org/.
Fall, A., Fall, J., 2001. A domain-specific language for models of landscape dynamics. Ecological Modelling 141, 1–18.
Gaff, H.D., Hartley, D.M., Leahy, N.P., 2007. An epidemiological model of Rift Valley Fever. Electronic Journal of Differential Equations 115, 1–12.
Gaucherel, C., Giboire, N., Viaud, V., Houet, T., Baudry, J., Burel, F., 2006. A domain-specific language for patchy landscape modelling: the Brittany agricultural mosaic as a case study. Ecological Modelling 194, 234–244.
Goodchild, M.F., 1992. Geographical data modelling. Computers & Geosciences 18, 401–408.
Langran, G., 1992. Time in Geographic Information Systems. London, Taylor & Francis.
Linard, C., Poncon, N., Fontenille, D., Lambin, E.F., 2009. A multi-agent simulation to assess the risk of malaria re-emergence in southern France. Ecological Modelling 220, 160–174.
Marvie, R., Pellegrini, M.-C., 2002. Modèles de composants, un état de l'art. Numéro spécial de L'Objet. Hermès-Lavoisier 8 (3), 61–89.
Maxwell, T., Costanza, R., 1997. A language for modular spatio-temporal simulation. Ecological Modelling 103, 105–113.
Mernik, M., Heering, J., Sloane, A.N., 2005. When and how to develop domain-specific languages. ACM Computing Surveys 37 (4), 316–344.
Murray, J.D., 2003. Mathematical Biology: I: An Introduction. Springer, pp. 79–83.
OMG, 2009. UML—Unified Modeling Language. http://www.uml.org.
Otero, M., Schweigmann, N., Solari, H.G., 2008. A stochastic spatial dynamical model for Aedes aegypti. Bulletin of Mathematical Biology 70, 1297–1325.
Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., 2007. Service-oriented computing: state of the art and research challenges. Computer 40 (11), 38–45.
Parent, C., Spaccapietra, S., Zimanyi, E., 2006. Conceptual Modelling For Traditional and Spatio-Temporal Applications: The MADS Approach. Springer-Verlag, Berlin, Heidelberg, 465 pp.
Peuquet, D.J., 1994. It's about time: a conceptual framework for the representation of temporal dynamics in geographic information systems. Annals of the Association of American Geographers 84 (3), 441–461.
Peuquet, D.J., 2001. Making space for time: issues in space-time data representation. Geoinformatica 5 (1), 11–32.
Ratzé, C., Gillet, F., Müller, J.-P., Stoffel, K., 2007. Simulation modelling of ecological hierarchies in constructive dynamical systems. Ecological Complexity 4, 13–25.
Szyperski, C., 1998. Component Software: Beyond Object-Oriented Programming. ACM Press and Addison-Wesley.
Tran, A., Raffy, M., 2006. On the dynamics of dengue epidemics from large-scale information. Theoretical Population Biology 69, 3–12.
Torrens, P.M., Benenson, I., 2005. Geographic automata systems. International Journal of Geographical Information Science 19 (4), 385–412.
Uhrmacher, A., Schattenberg, B., 1998. Agents in discrete event simulation. In: Proceedings of the European Symposium on Simulation (ESS'98, Nottingham, England, Oct.). Society for Computer Simulation, San Diego, CA.
W3C, 2002. WebServices. http://www.w3.org/2002/ws/.
Wachowicz, M., 1999. Object-Oriented Design for Temporal GIS. Taylor & Francis.
Wainer, G.A., Giambiasi, N., 2001. Cell-DEVS/GDEVS for complex continuous systems. Simulation 81 (2), 137–151.
Worboys, M.F., 1994. A unified model for spatial and temporal information. Computer Journal 37 (1), 26–34.
Wu, J., 1999. Hierarchy and scaling: extrapolating information along a scaling ladder. Canadian Journal of Remote Sensing 25, 367–380.
Yuan, M., 1999. Use of three-domain representation to enhance GIS support for complex spatiotemporal queries. Transactions in GIS 3 (2), 137–159.
Ziegler, B.P., Praehofer, H., Kim, T.G., 2000. Theory of Modeling and Simulation, 2nd ed. Academic Press, London.

Please cite this article in press as: Degenne, P., et al., Design of a Domain Specific Language for modelling processes in landscapes. Ecol. Model. (2009), doi:10.1016/j.ecolmodel.2009.06.018