

# Showcasing a Domain Specific Language for Spatial Simulation Scenarios with case studies

Luís de Sousa  
Luxembourg Institute of Science and Technology  
41, rue du Brill  
L-4422 Belvaux  
Luxembourg  
luis.a.de.sousa@gmail.com

Alberto Rodrigues da Silva  
IST / INESC-ID  
Rua Alves Redol, 9  
1000-029 Lisboa  
Portugal  
alberto.silva@tecnico.ulisboa.pt

## ABSTRACT

Cellular automata and agent-based modelling techniques have long been used for spatial simulation in the Geographic Information Systems field. However, they largely rely on code libraries and pre-compiled models, either requiring advanced programming skills or imposing scope constraints. Several domain specific languages have been proposed in this context, but mostly resulting in new textual programming languages.

DSL3S is a domain specific language for spatial simulation, synthesising concepts in a UML profile, permitting the design of simulation models through graphical elements. MDD3S is an implementation of this language relying on model-driven development (MDD) tools built around the Eclipse IDE; it produces ready to run simulations from DSL3S models, supported by the MASON simulation tool-kit. Additionally, this article shows the usage of DSL3S through a set of simple models that portrait classical constructs in spatial simulation.

## Keywords

Domain Specific Language, Spatial Simulation, UML Profile, Model-Driven Development

## 1. INTRODUCTION

In the Geographic Information Systems (GIS) domain, exploring how spatial variables and features evolve with time is often necessary. To this purpose several techniques have been developed, comprising a sub-domain of GIS referred as Spatial Simulation [de Smith et al., 2013]. They provide understanding on geographic phenomena in two essential ways [Batty, 2007]: (i) disclosing the dynamics that drove changes observed in the past and (ii) forecasting future changes.

The first code libraries integrating functionality dedicated to Spatial Simulation date back to the 1990s [Minar et al., 1996]. Many other tools followed, in a sprout of expansion around the turn of the century, today amounting to more than one hundred [de Smith et al., 2013]. However, they still pose important challenges to the spatial analyst, starting with a non trivial choice for the most suitable tool, plus the requirement for solid programming skills or in exchange the compromise of application scope.

Among the many Spatial Simulation tools available today two essential groups stand out: *Program-level* and *Model-level* tools [de Sousa and da Silva, 2011]. The first are

conceived for the programmer, mostly code libraries that encapsulate some of the complexity in specific methods or functions. Examples are Swarm [Iba, 2013], REPast [North et al., 2005] and MASON [Luke et al., 2005]. By their very nature, *Program-level* tools are not accessible to spatial analysts lacking programming skills and may require a long learning process. In contrast, *Model-level* tools, such as SLEUTH [Clarke et al., 1997], LANDIS [Mladenoff, 2004] or TELSA [Merzenich and Frid, 2005], are pre-programmed models that the user can parametrise, setting inputs and tuning pre-defined variables. They are easier to use, but also restrict the application scope; in some cases integration with spatial data is poor or non-existent.

Beyond these difficulties it has been recognised that an integrated approach to the description of agent-based models is largely lacking [Müller et al., 2014]. The reliance on source code or static documentation can create extra barriers when communicating model dynamics to stakeholders or peer analysts; model comparison and reuse are also difficult.

The *Domain Specific Language for Spatial Simulation Scenarios* (DSL3S) is a Domain Specific Language (DSL) for spatial simulation in the GIS context. It tries to ease the development of simulation models through a Model-Driven Development (MDD) approach, whereby models are developed through the arrangement of graphical elements and their relationships, dispensing formal programming knowledge. These graphical models can then be translated into ready to run simulations through the application of a code generation infrastructure [Selic, 2003].

The MDD approach raises the level of abstraction at which development takes place, thus simplifying the communication between programmers and analysts, and other stakeholders lacking programming skills [Mohagheghi et al., 2013]. It can also allow prototyping by non-programmers. By detaching model development from specific technologies, it can improve interoperability with geo-spatial data, generating *ad hoc* code as needed. Lastly, it can lay the foundations for a standard language in this domain, as successful efforts in parallel fields have proved, such as SysML<sup>1</sup> (for systems engineering) or ModelicaML<sup>2</sup> (for complex systems).

From the several specific MDD approaches available, the option rested on the Model-Driven Architecture (MDA)<sup>3</sup> proposed by the Object Management Group (OMG), that

<sup>1</sup><http://www.sysml.org/>

<sup>2</sup><https://www.openmodelica.org/index.php/home/tools/134>

<sup>3</sup><http://www.omg.org/mda/>

promotes UML profiles for the definition of DSLs. UML 2.0 allows the extension of its core primitives (graphical elements, links, etc) through specialisation for different application domains [OMG, 2005].

DSL3S takes spatial simulation as a branch of the wider Spatial Analysis GIS field, where model inputs primarily originate from a GIS and whose outputs also have geo-referenced relevance. At this time the language does not contemplate actors of change with internal cognitive capacities – commonly known as adaptive agents [Franklin and Grasser, 1997] – neither are considered any explicit concepts of society or societal interaction. All actors of change are assumed to exist in the space of simulation, thus forcefully being geographic entities. The language does not employ a distinction between agent based models and cellular automata, aiming at a single approach to both schools of spatial simulation, hiding such design and implementation details from the user.

This article exemplifies the usage of DSL3S through a set of use cases that portray its application to common spatial simulation scenarios. Section 2 reviews previous DSLs attempted in the field, noting how DSL3S differs. The abstract syntax of the language is briefly described in Section 3, as so MDD3S, its implementation framework. Use cases are described in Section 4; Section 5 summarises the article and discusses future work.

## 2. RELATED WORK

There have been several attempts to create DSLs for Spatial Simulation. They present ways to bridge the gap between *Program-level* and *Model-level* tools, approaching model description to natural language but still retaining some of the freedom of general purpose programming languages.

**NetLogo** is a specialisation of the Logo functional programming language, directed at Agent based simulations. It was initially an educational project to help students exploring emergent behaviour; progressively it evolved into a multi-platform tool with a Java based engine. A library of over 300 pre-built models has been gathered on-line<sup>4</sup>, that has powered wide adoption in academia. NetLogo is relatively easy to learn, especially when compared with code libraries, dispensing the skills required in object oriented programming [Railsback et al., 2006].

**SELES** (the Spatially Explicit Landscape Event Simulator) is a declarative DSL for Landscape Dynamics [Fall and Fall, 2001]. It was conceived to be used closely with geo-spatial data, supporting a vast range of different input raster formats (most common in Land Use / Land Cover data). It functions by interpreting a set of text files that declare variables, agents and events.

**MOBIDYC** (Modelling Based on Individuals for the Dynamics of Communities) is an agent based approach to the study of population dynamics, directed at the fields of Biology and Ecology [Ginot et al., 2002]. MOBIDYC is in essence a Smalltalk code package, defining a set of simple primitives, such as *environment*, *agent* and *state*; to these adds a set of pre-defined *behaviours*. These primitives are designed to make the language close enough to natural language, facilitating its usage by non-programmers. However, it has no support for the direct input of geo-spatial data.

**Ocelet** is a declarative DSL for landscape dynamics aimed at tackling common difficulties in capturing space-time dy-

namics with traditional modelling techniques [Degenne et al., 2009]. It takes an unconventional approach to this field by mimicking the concept of service-oriented architecture, with models built by components interacting with each other through services. The language is supported by two Eclipse plug-ins: a language editor and a code generator.

These DSLs focus mainly on providing a refined concrete syntax but still framed in older programming paradigms such as declarative or functional languages; a formal abstract syntax is usually absent. They still require the user to understand keywords and how to compose a coherent set of instructions or declarations into a specific model. Lack of interoperability with geo-spatial data is also an issue to some of these languages, as so platform or system dependency. In essence these efforts relying on textual languages fall into the same pitfalls identified by [Selic, 2008], regarding fourth generation languages: they struggle to hike the level of abstraction at which model development takes place.

A DSL not conceived for spatial simulation, but worthy of mention, is the Agent Modelling Language (**AML** [Trencansky and Cervenka, 2005], proposed for social dynamics. It is rather extensive, containing a wide range of UML stereotypes that use a large number of different UML meta-classes. Its concepts are organised hierarchically, through several levels of generalisation. No code generation infrastructure has ever been developed for AML and no applications could be found in the literature.

## 3. DSL3S: LANGUAGE AND TOOLS

This section provides a brief description of DSL3S and the prototype framework developed to support it. A more detailed account of the language can be found in [de Sousa and da Silva, 2012].

### 3.1 Language

Three main constructs can be identified underpinning a spatial simulation: Animats and Spatial and Global variables. **Animat** is a term coined by [Wilson, 1991], signifying *artificial animal*; in this context it is used more widely, representing all spatial elements that evolve themselves or induce change in their surroundings; examples are: fire (in a wildfire model), urban areas (in an urban sprawl model) or predators (in a population dynamics model). **Spatial** variables are spatial information layers that have some sort of impact on the dynamics of a simulation, e.g. slope that deters urban sprawl or biomass that feeds a wildfire. Beyond these key concepts, other elements can also be found in a simulation. **Global** variables may also exist, setting information that is constant across the space of simulation, such as capital stock in an urban sprawl model.

An Animat is composed by a set of **Attributes**, that describe its internal state at each moment in time. **Operations** make explicit the way animats act and react to the environment, thus encoding spatial dynamics. Six basic types of Animat operations are considered in DSL3S:

- **Emerge**: sets the probability or conditions under which a new animat can appear in the simulation space, e.g., the act of "birth". An example may be an urban development simulation where the emergence of new urban spots is possible in an area that meets a certain set of criteria, like distance to transport infrastructure or topography.

<sup>4</sup><http://ccl.northwestern.edu/netlogo>

- **Move**: relates an animat with spatial variables or with other animats, determining the locations that are more or less favourable to be in.
- **Replicate**: captures behaviours where an animat replicates itself, such as an organism in a biological simulation reproducing a sibling.
- **Supply**: provides access to the animat internal properties, thus supplying resources or information to other animats.
- **Harvest**: an act by which an animat may change other elements in its surroundings; it can act on a spatial variable, such as a wildfire consuming bush, or by seizing resources from another animat, as in a predator-prey simulation.
- **Perish**: defines the circumstances under which an animat may cease to exist during simulation; examples can be a biological animal starving or a fire extinguishing.

This strict set of operations tries to match the essential properties of an agent, as outlined by [Franklin and Grasser, 1997]: (*autonomous, continuous, reactive, proactive* and *mobile* with the core concepts found in Cellular Automata (*state, neighbourhood, transition rules* and *time*). In their seminal work [Epstein and Axtell, 1996] conceive a considerably larger set of operations, including elaborate processes such as trade and cultural exchange. DSL3S restricts its set of operations for three reasons: (i) because such refined operations are less common in spatial simulation applications and can eventually be composed with simpler primitives; (ii) to keep the language compact and easy to learn; and (iii) to insulate the user from the technical implementation details in the choice between Cellular Automata and Agent based models.

Figure 1 presents these key constructs in a conceptual model. Each of these constructs is realised by a specific stereotype in the DSL3S UML profile; a stereotype exists also for each operation type. A Simulation is composed by a set of Animats, Spatial and Global variables; Animats are composed by a set of Attributes and Operations, that determine how their internal state evolves. An animat acts through different types of Operations, that can induce changes on spatial variables or the state of other animats.

## 3.2 Tool Support

*Model Driven Development for Spatial Simulation Scenarios* (MDD3S) is the name of the prototype framework developed to support the DSL3S language. MDD3S relies solely on open source tools: (i) **Papyrus** - an Eclipse<sup>5</sup> add-on for UML modelling; (ii) **Acceleo** - another Eclipse add-on supporting model-to-code generation templates; (iii) **MASON** - a *Program-level* spatial simulation framework used as a library by the code generated. The full software stack is described in Figure 2.

Papyrus is a graphical editor for the UML language based on the Eclipse Modelling Framework<sup>6</sup> (EMF). It allows the edition and visualisation of structured models defined with

<sup>5</sup><http://www.eclipse.org/modeling>

<sup>6</sup><http://www.eclipse.org/modeling/emf/>

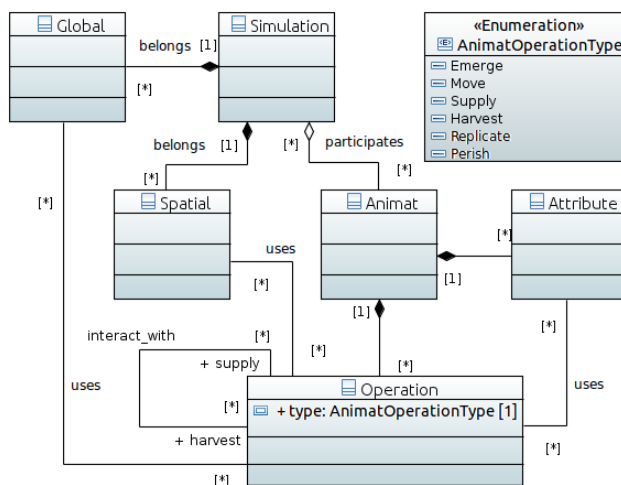


Figure 1: The DSL3S meta-model.

the XMI language, providing a set of Java classes that facilitate its manipulation. Papyrus evolved to support the development of *ad hoc* DSLs, through the definition of UML profiles. It is presently close to fully support version 2 of the UML language.

Acceleo<sup>7</sup> is an open source code generator also built on EMF. Acceleo interprets templates written with the MOF Model to Text Transformation Language<sup>8</sup> (MOFM2T), also an OMG standard. It fully supports code generation from meta-models, identifying stereotypes applied on classes and providing access to its properties. The later is not based on MOFM2T, rather provided by a special service – essentially a user developed Java method that browses through the UML2 object model associated with each class.

MASON (acronym for “Multi-Agent Simulator Of Neighbourhoods”) aims to be a light-weight, highly portable, multi-purpose agent-based modelling package [Luke et al., 2005]. It is fully written in Java and open source, and is likely the most performant of the main *Program-level* tools for spatial simulation [Railsback et al., 2006]. GeoMASON<sup>9</sup> is an extension that provides Java objects to deal specifically with geo-referenced data. Input and output functionality is available for both raster and vector datasets, relying on third party packages: the Java Topology Suite<sup>10</sup>, GeoTools<sup>11</sup> for vector input and output and GDAL<sup>12</sup> for raster formats.

<sup>7</sup><http://www.acceleo.org/pages/introduction/en>

<sup>8</sup><http://www.omg.org/spec/MOFM2T/1.0/>

<sup>9</sup><http://cs.gmu.edu/eclab/projects/mason/extensions/geomason/>

<sup>10</sup><http://www.vividsolutions.com/JTS/JTSHome.htm>

<sup>11</sup><http://www.geotools.org/>

<sup>12</sup><http://www.gdal.org/>

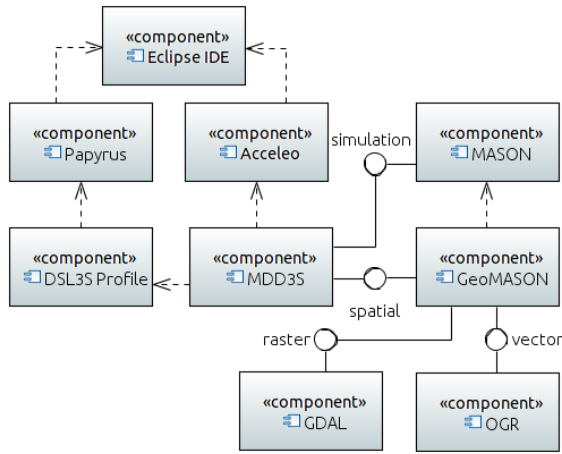


Figure 2: The technologies used to implement MDD3S.

## 4. CASE STUDIES

The DSL3S UML profile and its accompanying MDD3S framework are publicly available at the code sharing platform GitHub<sup>13</sup>. Some examples are also available to showcase the usage of the language that can be accessed with Papyrus or any other software able to interpret the XMI language. In this section three of these simulation models are discussed.

### 4.1 Simulation Model A – Urban Sprawl

The study of urban dynamics was one of the first applications of spatial simulation techniques. The growth of cities is taken generally as an emergent process, by which the urban fabric sprawls, bounded by spatial restrictions and enablers. SLEUTH [Clarke et al., 1997], a *Model-level* tool dating back to the 1990s, proved particularly successful in this domain and has been applied to varied geographic contexts.

#### 4.1.1 The DSL3S model

In this example space is vacant at simulation start and is progressively occupied by urban elements. The simulation is built around four elements: (i) an **Animat** named *Urbe* that possesses a single **Attribute**, storing its *Age*; (ii) a **Global** variable termed *Speed* that declines with time; (iii) a **Spatial** element to input a vector layer with *Protected* areas, where urban growth is not possible; (iv) another **Spatial** variable that inputs a *Roads* layer, an enabler of urban sprawl. Figure 3 presents the full model in three views.

At simulation start a single *Urbe* animat is cast at random in the simulation space; it is not mobile, with the dynamics defined through **Emerge** operations. In first place there is the *SprawlAge* operation that sets the probability of a new *Urbe* animat emerging nearby an existing *Urbe*; this operation is linked to the *Age* attribute, to render emergence less probable near older urban areas. Also to constraint growth with time (mimicking diminishing capital investment) is the *SprawlSpeed* operation, linking to *Speed* – as its internal value declines with time, it slows down growth. *SprawlRoads* relates *Urbe* with *Roads*, increasing the probability of emergence around spatial features of this layer. In similar fashion, *SprawlProtect* sets the probability of emergence to

zero (with a large negative weight) over spatial features in the *Protected* areas layer. The *Simulation* space is set to a grid of 100 by 100 cells, this way determining the space between adjacent emerging urban elements.

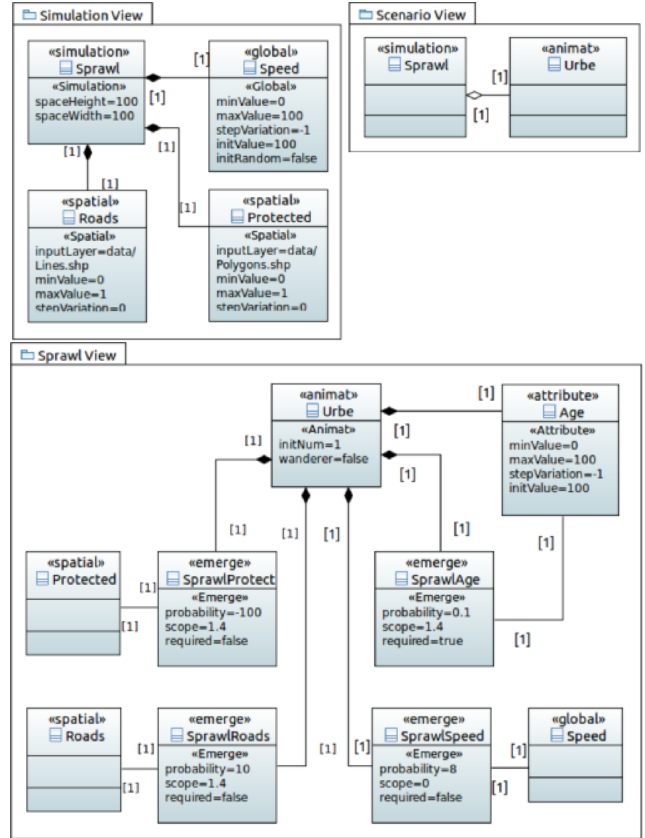


Figure 3: Urban Sprawl model in DSL3S.

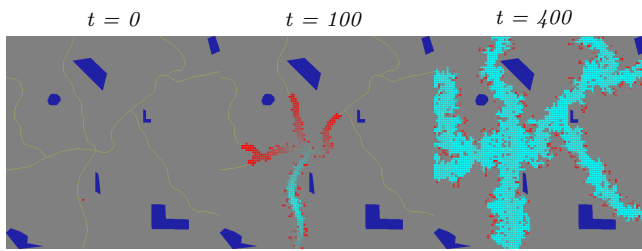
#### 4.1.2 The resulting application

Figure 4 shows the simulation space resulting from this model at time steps 0, 100 and 400. In the very beginning there is a single urban element, presented in red; in dark blue are represented *Protected* areas, while yellow lines portrait *Roads*. Development is fast in the beginning, with new urban elements emerging along road features; as they age, the colour of *Urbe* elements slowly fades to a light cyan. With time sprawl slows down and a larger number of steps is required for changes in the urban fabric to become apparent. With the areas surrounding *Roads* taken, sprawl then turns inwards, but avoiding *Protected* polygons.

### 4.2 Simulation Model B – Predator-Prey

Predator-Prey simulations are one of the oldest applications of spatial simulation techniques [Dewdney, 1988], used to study population dynamics in the field of Biology. It usually features two animal species, where one feeds off the other; energy flows through the food chain in waves whose period and amplitude are function of the growth rates of the several species.

<sup>13</sup><https://github.com/MDDLingo/DSL3S>



**Figure 4:** A sample run of the Urban Sprawl DSL3S simulation; *Urbe* is portrayed with a red to cyan ramp, *Roads* are portrayed in yellow and *Protected* areas in dark blue.

#### 4.2.1 The DSL3S model

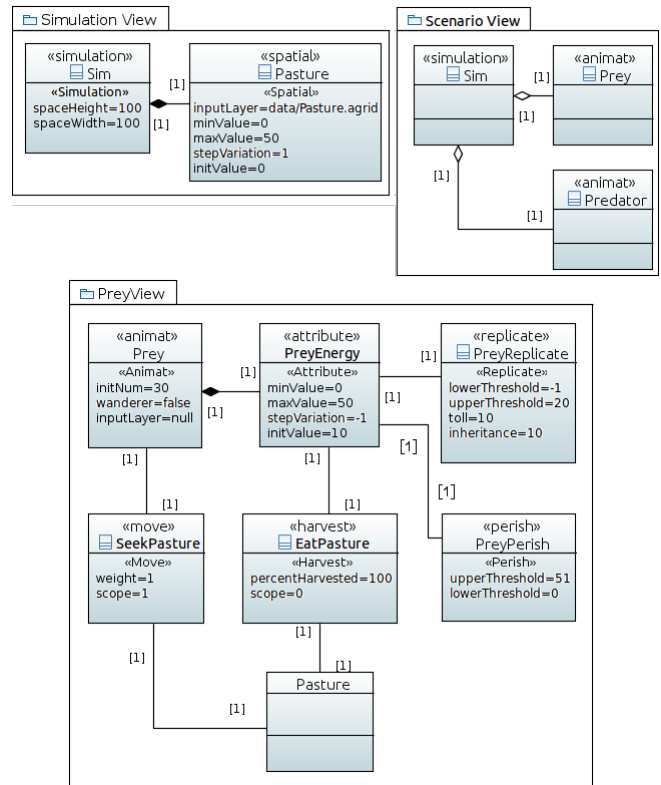
This example takes place in a synthetic plane of 100 by 100 abstract space units. There are three main elements to this simulation: a **Spatial** variable named *Pasture* and two animats: *Predator* and *Prey*. *Pasture* covers the whole simulation space and is initiated from a sample raster file that represents energy available at each space unit. This energy at each location increases at each time step, at a fixed rate during simulation, up to a defined limit.

*Prey* is a herbivore animat composed by a single **Attribute**: *Energy*. At simulation start a number of these animats are cast randomly across the simulation space, with its *Energy* state also randomly initialised. *Energy* declines steadily at each time step by a defined amount. A **Perish** operation attached to *Energy* sets a lower threshold below which the animat is discarded from the simulation. A **Supply** operation linked to *Energy* makes this attribute available for the *Predator* animat. A **Harvest** operation parametrises the feeding act of *Prey* over *Pasture*; at each time step the animat can take all the *Pasture* energy available at the location it occupies into its own *Energy* state. Two **Move** operations relate *Prey* with both *Pasture* and *Predator*, making it prefer locations with high *Pasture* energy and free of *Predator* instances. Finally, a **Replicate** operation sets a threshold above which the *Prey* can reproduce itself, as so the amount of energy passed on to the offspring in the process (the DSL3S view for *Prey* is shown in Figure 5).

*Predator* is a carnivore animat that shares many similarities with *Prey* (Figure 6). It also possesses a single state (*Energy*) and its instances are created from an input vector layer, using its attributes for state initialisation. Its state also declines with time and a **Perish** operation determines when it ceases to exist. *Predator* feeds on *Prey*, according to an **Harvest** operation linking to the **Supply** operation of *Prey*. When a *Predator* feeds off a *Prey* it takes up all of its energy, triggering its **Perish** operation. A single **Move** operation links *Predator* to the *Prey* energy state, this way compelling it to move towards locations where well nurtured *Prey* instances exist. A **Replicate** operation sets similar reproduction conditions to those for *Prey*.

#### 4.2.2 The resulting application

The simulation generated from this model produces the typical population cycles seen in this type of models, as in the case of the historical WATOR model [Dewdney, 1988]. Figure 7 shows the simulation space during a sample run at



**Figure 5:** Predator-Prey model in DSL3S; Simulation, Scenario and Prey Views.

time-steps 0, 30, and 90. *Prey* animats reproduce faster and thus dominate space during the first time steps, producing an initial wave reaping the fertile feed stock. In time, *Predator* animats feed off the excessive amount of *Prey* animats creating a new wave; this *Predator* wave clears some areas, fostering growth of *Pasture* in certain patches.

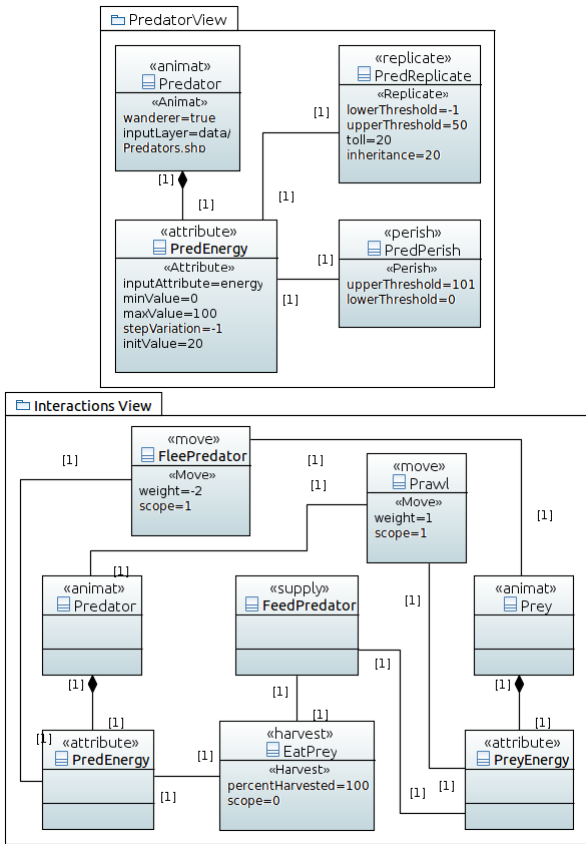
### 4.3 Simulation Model C – Game of Life

The Game of Life [Gardener, 1970] is a cellular automata simulation that had a capital role in popularising spatial simulation techniques. It does not have any direct real world application, but showcases the emergence of complex patterns in a system with very simple rules.

#### 4.3.1 The DSL3S model

Implementing the Game of Life with DSL3S is an interesting exercise since this simulation is composed of a single static element, the *Life* animat, from which all operations must be derived. This animat is composed by two attributes, *Alive*, a binary variable that indicates if the animat is dead or alive, and *Neighbours*, that stores the amount of living animats in its neighbourhood. The *Life* animat makes its state known through a **Supply** operation, a linked **Harvest** operation is used to count the number of neighbours at each time step. Having no impact on the harvested animats, this setting exemplifies the usage of **Harvest** for mere information collection.

The rules of this famous simulation are implemented using in first place an **Emerge** operation linked to the *Alive* attribute. Using a scope encompassing the immediate neigh-



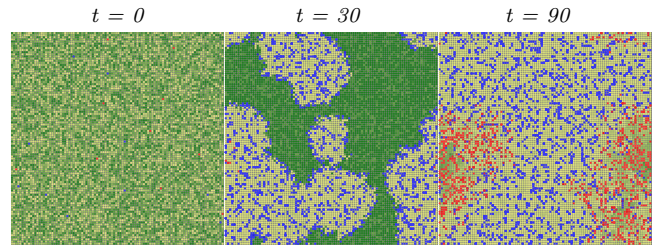
**Figure 6: Predator-Prey model in DSL3S; Predator and Interaction Views.**

bourhood of a location, the resulting **Emerge** class counts the number of animats in the surroundings and creates a new **Life** animat if this number is three. The death of a **Life** instance is defined with a **Perish** operation associated with the **Neighbours** attribute, setting an admissible interval between two and three neighbours for the animat to survive. The **Neighbours** attribute is reset to its minimum at the beginning of each time step to force the re-evaluation of the survival conditions for the **Life** animat (figure 8 presents the Life model described in DSL3S).

#### 4.3.2 The resulting application

The definition of this historical model with DSL3S may not be the most obvious – even though fitting in three small diagrams – but serves to prove that a simulation like the Game of Life can be implemented with the language. In the model available at GitHub some classical initial configurations are provided in a vector layer to test this example, but by default the simulation space is initialised randomly, providing for some interesting evolutions from chaos into repetitive patterns.

Figure 9 shows snapshots of a simulation run with 1 000 random locations set as alive at time-step 0. After 50 time-steps stable structures already start appearing, but chaotic patches can still grow or move in ways to disturb surrounding spaces. At time-step 200 there are still large chaotic patches and some stable structures that emerged previously have been destroyed; at this time the emergence of *glider*



**Figure 7: A sample run of the Predator Prey DSL3S simulation; Pasture is portrayed with a yellow to green colour ramp, Prey is portrayed in blue and Predator in red.**

elements is still common. With an initial random configuration as this, it can take in the order of 500 time-steps for the entire space to stabilise.

## 4.4 Discussion

These three case studies show how DSL3S elements can be combined to produce diverse simulations on different fields of application. An Urban sprawl model exemplifies the development of non mobile agents that emerge over the simulation space; it demonstrates how to combine different input spatial layers to influence Animat behaviour. A Predator-Prey model portrays other classical spatial dynamics elements: mobile agents that move towards and away spatial objects and perform changes to the landscape and other agents. Finally, a realisation of the classical Game Of Life cellular automata demonstrates how DSL3S constructs can be composed to create models with less obvious dynamics.

Table 1 presents basic statistics on the source code generated. The Urban Sprawl model, that is relatively complex, with one **Global** and two **Spatial** elements, results in over 1 000 lines of code; this from a DSL3S model with just three views, that fit in a single picture. The Predator-Prey model produces less classes but more code, since it includes more, and more diverse, operations. If for the first two examples the amount of code generated may indicate that much complexity is represented at model level, the Game of Life model hints at an exaggerated amount of code generated. Even though this particular model realisation takes in geo-spatial data and sets up a graphical interface, it can be argued that an *ad hoc* implementation with an object oriented language would be leaner. This may be one of the short-comings of the MDD approach, even if the user is not expected to work directly on the source code.

**Table 1: Statistics on the code generated.**

Model	Number of classes	Lines of code
Urban Sprawl	11	1075
Predator Prey	10	1192
Game of Life	7	757

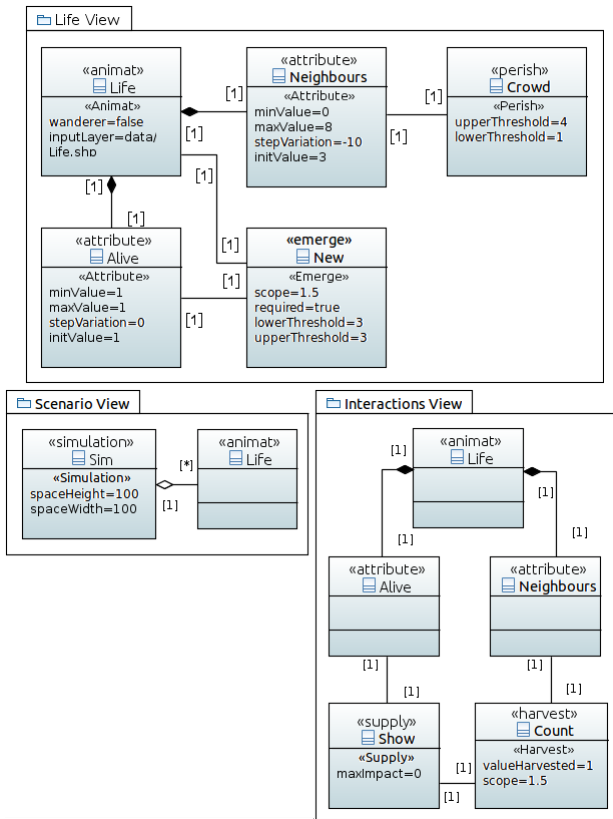


Figure 8: The Game of Life model in DSL3S.

The Game of Life model also points to eventual difficulties with the relatively strict size of the language. Although DSL3S is able to represent and generate this sort of simulation, the graphical model may not be straightforward. The employment of the language may thus require substantial guidance in the form of a manual or other methodological support documents.

## 5. CONCLUSION AND FUTURE WORK

The application of spatial simulation techniques to the GIS domain remains today locked in the choice between versatile tools, that require advanced programming skills, and the option for ease of use with pre-built models, in such case implying relevant compromises of transparency and scope. Several DSLs, and respective tools, such as NetLogo, SELES or MOBIDYC, were tried in this field, but invariably producing declarative or functional languages, in some cases lacking a formal abstract syntax. They also impose compromises with platform dependence and in some cases with weak GIS support and interoperability.

DSL3S proposes an MDD approach to this subject, underpinned by an UML profile that forms a graphical language. MDD3S is a prototype framework that implements this language, composed by a modelling designer and a model-to-code transformation infrastructure. These assets are able to translate a graphical, abstract and platform independent model produced with DSL3S into a coded simulation supported by a *Program-level* tool. In other applications, the employment of MDD methodologies has proved capable of inducing faster development [Clark and Muller, 2012], re-

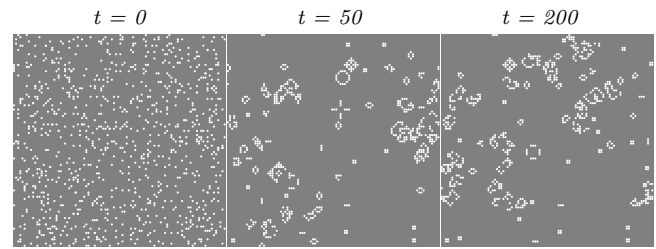


Figure 9: A sample run of the Life DSL3S simulation, with living cells portrayed in white.

duce coding errors [Paige and Varró, 2012] and improve model readability [Mohagheghi et al., 2013].

The MDD3S framework currently relies on MASON, a modern Java library for spatial simulation. This option also guarantees interoperability with geographic data, namely through the GeoMASON extension. This framework is being developed on the Eclipse IDE, using the MDD ad-ons Papyrus, for UML modelling, and Aceleo, for code generation. Both the DSL3S UML profile, the MDD3S framework and the example models presented are in the public domain.

DSL3S will be further assessed through its application to real world scenarios. An iterative process shall provide an understanding of how far it can go in its current form and if extensions are necessary. Graphical semantics is another area where improvements are possible, in particular through the employment of stereotype icons proposed before [de Sousa and da Silva, 2012]. Such feature is not presently supported by the MDD tools used, but hopefully will be available in the following release of the Eclipse software stack.

A series of support contents has been produced to facilitate the first contact with the language<sup>14</sup>. These contents are presently being used in an evaluation experience<sup>15</sup> where international spatial simulation experts and environmental and computer sciences professional are invited to test the language and feed back on its usability and productivity.

## 6. REFERENCES

- [Batty, 2007] Batty, M. (2007). *Cities and Complexity*. MIT Press.
- [Clark and Muller, 2012] Clark, A. and Muller, P. (2012). Exploiting model driven technology: a tale of two startups. *Software & Systems Modeling*, 11(4):481–493.
- [Clarke et al., 1997] Clarke, K., Hoppen, S., and Gaydos, L. (1997). A self-modifying cellular automaton model of historical urbanization in the san francisco bay area. *Environment and Planning B*, 24:247–261.
- [de Smith et al., 2013] de Smith, M. J., Goodchild, M. F., and Longley, P. A. (2013). *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools - Fourth Edition*, chapter Geosimulation, pages 625–672. Winchelsea Press.
- [de Sousa and da Silva, 2011] de Sousa, L. and da Silva, A. R. (2011). Review of spatial simulation tools for geographic information systems. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind.

<sup>14</sup><https://github.com/MDDLingo/DSL3S/wiki>

<sup>15</sup><https://github.com/MDDLingo/DSL3S/wiki/Evaluation>

- [de Sousa and da Silva, 2012] de Sousa, L. and da Silva, A. R. (2012). Preliminary design and implementation of dsl3s - a domain specific language for spatial simulation scenarios. In *Proceedings of CAMUSS - Cellular Automata Modelling for Urban and Spatial Systems*, pages 269–280. Dep. of Civil Engineering of the Univ. of Coimbra.
- [Degenne et al., 2009] Degenne, P., Lo Seen, D., Parigot, D., Forax, R., Tran, A., Ait Lahcen, A., Cur, O., and Jeansoulin, R. (2009). Design of a Domain Specific Language for modelling processes in landscapes. *Ecological Modelling*, 220:3527–3535.
- [Dewdney, 1988] Dewdney, A. K. (1988). *The Armchair Universe*, chapter Sharks and Fish on the Planet Wa-Tor, pages 239–251. W. H. Freeman, New York.
- [Epstein and Axtell, 1996] Epstein, J. M. and Axtell, R. (1996). *Growing Artificial Societies*. MIT Press.
- [Fall and Fall, 2001] Fall, A. and Fall, J. (2001). A domain-specific language for models of landscape dynamics. *Ecological Modelling*, 141:1–18.
- [Franklin and Grasser, 1997] Franklin, S. and Grasser, A. (1997). Is it an agent or just a program? a taxonomy for autonomous agents. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 21–35, Berlin. Springer-Verlag.
- [Gardener, 1970] Gardener, M. (1970). Mathematical Games - The fantastic combinations of John Conway’s new solitary game ‘life’. *Sicentific American*, October:120–123.
- [Ginot et al., 2002] Ginot, V., Le Page, C., and Souissi, S. (2002). A multi-agents architecture to enhance end-user individual based modelling. *Ecological Modelling*, 157:23–41.
- [Iba, 2013] Iba, H. (2013). *Agent-Based Modeling and Simulation with Swarm*. Chapman and Hall/CRC.
- [Luke et al., 2005] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005). Mason: A multi-agent simulation environment. *Simulation: Transactions of the society for Modeling and Simulation International*, 82(7):517–527.
- [Merzenich and Frid, 2005] Merzenich, J. and Frid, L. (2005). Projecting Landscape Conditions in Southern Utah Using VDDT. In Bevers, M. and Barrett, T. M., editors, *Systems Analysis in Forest Resources: Proceedings of the 2003 Symposium*, pages 157–163, Portland, OR. U.S. Department of Agriculture, Forest Service, Pacific Northwest Research Station.
- [Minar et al., 1996] Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996). *The Swarm simulation system: a toolkit for building multi-agent simulations*.
- [Mladenoff, 2004] Mladenoff, D. (2004). LANDIS and forest landscape models. *Ecological Modelling*, 180(1):7 – 19.
- [Mohagheghi et al., 2013] Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M., Nordmoen, B., and Fritzsche, M. (2013). Where does model-driven engineering help? Experiences from three industrial cases. *Software & Systems Modeling*, 12(3):619–639.
- [Müller et al., 2014] Müller, B., Balbi, S., Buchmann, C. M., de Sousa, L., Dressler, G., Groeneveld, J., Klassert, C. J., Le, Q. B., Millington, J. D., Nolzen, H., Parker, D. C., Polhill, J. G., Schlüter, M., Schulze, J., Schwarz, N., Sun, Z., Taillandier, P., and Weise, H. (2014). Standardised and transparent model descriptions for agent-based models: Current status and prospects. *Environmental Modelling & Software*, 55(0):156 – 163.
- [North et al., 2005] North, M. J., Howe, T. R., Collier, N. T., and Vos, J. R. (2005). The Repast Symphony development environment. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, Chicago, USA.
- [OMG, 2005] OMG (2005). UML 2.0 Specification. <http://www.omg.org/spec/UML/2.0/>. Retrieved January 2011.
- [Paige and Varró, 2012] Paige, R. F. and Varró, D. (2012). Lessons learned from building model-driven development tools. *Software & Systems Modeling*, 11(4):527–539.
- [Railsback et al., 2006] Railsback, S. F., Steven, L. L., and Jackson, J. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623.
- [Selic, 2003] Selic, B. (2003). The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25.
- [Selic, 2008] Selic, B. (2008). Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, 15(3-4):379–391.
- [Trencansky and Cervenka, 2005] Trencansky, I. and Cervenka, R. (2005). Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS. *Informatica*, 29:391–400.
- [Wilson, 1991] Wilson, S. W. (1991). The animat path to AI. In Meyer, J. A. and Wilson, S., editors, *From Animals to Animats*, pages 15–21, Cambridge, MA. MIT Press.