Preliminary Design and Implementation of DSL3S – a Domain Specific Language for Spatial Simulation Scenarios

Luís DE SOUSA[‡]; Alberto Rodrigues DA SILVA[ª]

[‡]Resource Centre for Environmental Technologies, Public Research Centre Henri Tudor

66 Rue de Luxembourg, Esch-sur-Alzette, L-4221 Luxembourg

+352 42 59 91 3356, luis.a.de.sousa@gmail.com

[ª]Instituto Superior Técnico

Lisbon, Portugal

alberto.silva@acm.org

Keywords: spatial simulation; domain specific language; model driven engineering; cellular automata; agent based modelling

Introduction

Spatial Simulation in the context of Geographic Information Systems (GIS) is used primarily to assess the evolution of spatial variables over time. Since the 1990s two main techniques have been applied to perform this sort of spatial analysis: Cellular Automata [1] and Agent-based modelling [2]. The simulation models produced with these methods tend to be quite specific, only usable within the particular field of application, largely due to the multi-dimensional and heterogeneous character of spatial data. Mainly for this reason, modern multi-purpose GIS packages, such as GRASS[1] or ArcGIS[2], largely lack tools dedicated to this technology.

Developing a spatial simulation model using a general purpose programming language presents several burdens. Besides implementing the model, the program has to control the flow of execution, manage system resources, and manipulate data structures. This leads to several problems [3]: (i) difficulties verifying correct implementation; (ii) limited model generality due to difficult modification and/or adaptation; (iii) difficulty comparing computer models, usually restricted to their inputs and outputs [4]; (iv) problematic integration with other models or tools (e.g. GIS or visualisation packages).

Beyond general purpose programming languages, a spectrum of Spatial Simulation tools exists, ranging from code libraries (referred to as Program-level tools), that support the coding activity, to pre-built models (or Model-level tools), that can be

1 http://grass.osgeo.org/

2 http://www.arcgis.com/

used by simple parametrisation [3] (see Figure 1). Somewhere in the middle of this spectrum lay Domain Specific Languages (DSL).

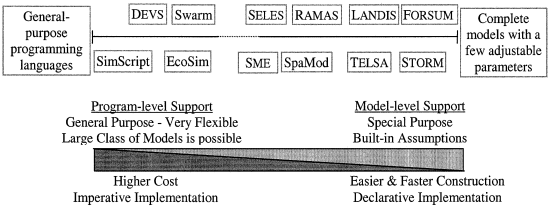

Figure 1: The Spatial Simulation tools spectrum devised by Fall and Fall [3].

In spite of this wide spectrum, spatial analysts are still constrained by the trade off between the burdens of programming and the strictness of pre-built models; with existing DSL not yet exactly freeing the analyst from coding.

This article presents a different approach to spatial simulation in the GIS domain, through the employment of standards from the Object Management Group[3] (OMG) to produce a graphical DSL. Starting is a short review of the types of simulation tools available today, their strengths and shortcomings; then, present difficulties are identified and the approach stated. The language is then presented in conceptual terms and closing its implementation is detailed.

Spatial simulation tools

As stated in the Introduction, three main categories of tools can be devised: Program-level, Model-level and DSL-level. This section describes each and briefly discusses its main characteristics. A more detail review is provided by de Sousa and da Silva [5].

Program-level support tools extend the facilities available in general-purpose programming languages, usually providing code libraries for building specific classes of simulation models. Examples of these tools are Swarm[4], RePAST [6] and MASON [7]. Higher-level code, usually in a general-purpose object-oriented programming language, specifies how objects are used to produce the desired behaviour. The main advantage of this type of tools is the encapsulation of the model from functionality, relieving the modeller from banal programming tasks and potentially producing leaner and easier to read code. These tools are tendentiously open source, operational on several computer platforms and providing good level of integration with GIS packages. Coupling this characteristic to their wider application scope, Program-level tools usually gather around them large communities of users,

3 http://www.omg.org/

4 http://www.swarm.org/index.php/Main_Page

that provide informal, but extensive, support. On the downside, these tools require an extra learning effort for their proper use. Beyond requiring relevant knowledge on the programming language itself [8], the modeller must learn to some detail the behaviour of functions, objects and methods provided by the tool kit, something that may require several months of practice [9].

Model-level support tools allow the employment of spatial simulation without requiring programming. These tools provide pre-programmed simulatio models, designed for specific application fields that can be parametrised by the end user. They provide fairly straightforward and rapid mechanisms for implementation, but invariably constraint the modeller to a specific application framework. Examples of such tools are SLEUTH [10], TELSA [11] and AnyLogic[5]. These tools tend to be quite specific, and much of the model behaviour and assumptions are hidden by the framework; their use in other application fields is largely impossible. They also tend to lack GIS interoperability, in best cases requiring specific data formats. Traditionally, they take advantage of market niches providing for the needs of a restricted group of users. Thus, in most cases, they are commercial products and their users community tend to be weak or non-existent, more often support is a paid service.

Midway between Program-level and Model-level are DSL-level tools, that provide a specific language specialised for a simulation domain. Compared to Model-level tools, these languages make fewer assumptions about the underlying simulation model structure. Examples include NetLogo[6], SELES [3] and MOBIDYC [12]. The use of DSL facilitates modelling and reduces the build-up time of simulation models. The programming environment is more constrained that in Program-level tools, with behaviour described using simple constructs. Still, the user has to understand keyword meaning and how to compose a set of instructions into a program. In general this category of tools doesn't provide much support for GIS integration, some even totally lacking such functionality. Users communities tend to be larger than those of Model-level tools, but on the other hand platform dependency is often an issue.

Difficulties and Approach

When using a tool for spatial simulation a GIS analyst is faced with some important challenges, namelly:

- Most spatial simulation tools require advanced programming skills;
- Those that do not require such knowledge are narrow scoped and invariably lack GIS interoperability;
- There's no standard or common language to describe spatial simulation models.

5 http://www.xjtek.com/anylogic/why_anylogic/

6 http://ccl.northwestern.edu/netlogo

Analysts working with spatial data either come from GIS related areas, like Geography or Geodesy, or from the scientific areas of application, such as Biology or Economics. In spatial simulation projects is almost mandatory the involvement of programmers, skilled in advanced methods like object-oriented technologies. This creates a further communication step from model concept to its implementation.

The adoption of pre-compiled Model-level tools also imposes its burdens. The correct implementation of such models is often hard or impossible to verify, since most are commercial, or otherwise closed source tools. Their static structure imposes strict compliance to their conceptual framework. Experiments with different behaviours or the input of alternative spatial information is impossible.

A simulation model that can only be described by the underlying source code becomes inaccessible to most GIS analysts. Source code specificities, such as data input/output or control structures, produce a layer of obfuscation that complicates the comparison of different models. There are a number of concepts that are common to any spatial simulation, such as the succession of time, spatial variables, agents or spatial location, but two implementations of a same model can appear entirely different if based on different tools, that impose different software architectures.

Our work attempts to address these issues through the development of the DSL3S, a Domain Specific Language for Spatial Simulation Scenarios. This language takes spatial simulation as a branch of the wider Spatial Analysis GIS field, where model inputs originate at least partially from a GIS and whose outputs may also have geo-referenced relevance. The same approach shall be taken to models that are traditionally implemented with cellular automata and to those based on agents, seeking a language abstracted from such technical differences.

DSL3S is design as a UML profile[7] because it allows the development of simulation models through UML class diagrams (to which stereotypes from the profile are applied and parametrised with properties). These models are then feed to a model-to-code transformation facility producing a ready-to-run simulation model on top of a Program-level tool. Then the analyst can chose to either perform tuning at model level or further refine the corresponding source code with the assistance of a developer.

The improvements with this approach are:

- *Faster development*, reducing the lag from prototype conception to testing;
- *Reduction of errors*, by reducing (or even eliminating) coding activities;
- *Increased readability*, with models described by graphical diagrams;
- *Improved GIS interoperability,* by using the modern Program-level tools;

7 http://www.omg.org/technology/documents/profile_catalog.htm

- *One model, several implementations*, code generation templates can be developed for different target Program-level tools.

The following section describes the concepts underlying the DSL3S.

The DSL3S Meta-model

Three main pillars have been identified as the underpinning concepts of a spatial simulation: Spatial Environment Variables, Animats and Behaviour. **Spatial Environment Variables** are spatial information layers that have some sort of impact on the dynamics of a simulation, e.g. slope that deters urban sprawl, biomass that feeds a wildfire. **Animat** is a term coined by S. W. Wilson [13] signifying *artificial animal*; in the context of DSL3S it is used more widely, representing all types of agents of change, such as wildfires, urban areas or agents in a predator-prey model. **Behaviour** associates the former two concepts, defining how Animats react to their surrounding environment and internal state, examples can be movement or replication.

Beyond these core concepts, other elements can also be found in a simulation, particularly context variables. **Global Variables** define information that is constant across the whole space of simulation, such as wind direction in a wildfire model. Global variables can eventually change with time, simulating changing environment conditions (again wind direction is a good example). Animats can have variables themselves, called **State Variables**, that detail their internal state.

A **Simulation** is composed by a set of Spatial Environment variables, Animats and Global variables; Animats may be composed by series of State variables. Animats are also composed by Behaviours that determine how it's internal state evolves; behaviours can be function of global variables, spatial environment or the state of other animats (see Figure 2). All these concepts are stereotypes in the DSL3S profile (see Figure 3).

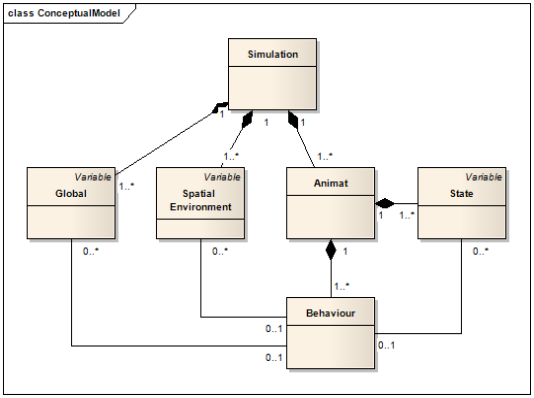

Figure 2: The core concepts of DSL3S.

The Global Variable is intended to be a simple scalar value that may vary with time. It can for instance be set randomly at simulation start and/or made to vary randomly each time step; it can also be fed into the model as a pre defined time-series of values, for instance read from an input text file.

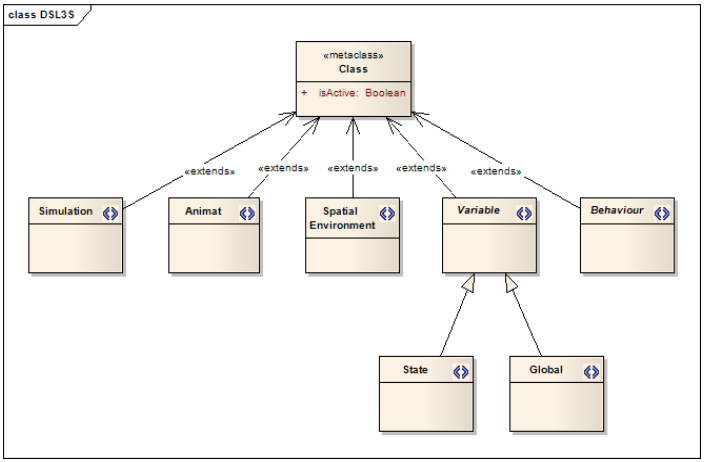

Figure 3: The main stereotypes of DSL3S.

The stereotype for Spatial Environment Variable is essentially a stub for the input of geo-referenced data. Each instance shall correspond to a spatial layer with the characteristic of having an unequivocal value for each location in space. No reference system is made explicit, it is assumed that all spacial data imputed to the simulation is bound to the same system and the model effectively operates in an ordinary Cartesian plane. For now no distinction between vector and raster data is being considered in the meta-model.

Animats are essentially an aggregation of state variables residing at a perfectly identifiable location in space. Different types of Animat can model specific genres of actors, e.g. wolves and sheep in a predator-prey simulation. The initial number of animat instances and their spatial positioning can be provided by a specific geo-referenced data set such as a raster map. By similar mechanisms the initial values of state variables can too be set with geo-referenced data.

The elements presented so far focused on retaining the information needed to run a spatial simulation. But more than that is required to capture spatial dynamics, the way animats act has to be made explicit. In DSL3S this aspect is modelled with specialisations of Behaviour, more precisely: **Initiate**, **Move**, **Replicate**, **Harvest** and **Perish**.

Initiate captures the conditions under which a new instance of an animat can appear in the simulation, the act of "birth". An example may be an urban development simulation where the emergence of a new urban spot is possible in an area that meets

a certain set of criteria like distance to transport infrastructure or topography. Probabilities of birth can also be defined with properties.

Move relates an animat with one or more spatial environment variables or other animats, determining the locations that are more or less favourable to be in. Specific properties allow to weight the relevance of each class related to a Move behaviour. For instance, in a predator-prey simulation the movement of a "sheep" animat may be positively weighted in a relationship with a "grass" class and negatively weighted in a relation with a "wolf" animat.

Replicate captures behaviours where an animat replicates itself to an adjacent location, such as a wildfire spreading or an urban area sprawling. Just as with previous behaviours, the objective is to capture the conditions under which an animat may originate a sibling into its neighbourhood. Properties may weight the influence of spatial environment variables (e.g. fire spread) or set thresholds against the animat's internal state (e.g. biological reproduction).

Harvest an act on which an animat may change other elements in the same spatial location; it can act on a spatial environment variable, such as a wildfire consuming bush, or by seizing another animat as in a predator-prey simulation. In this class properties parametrise the changes of this action on both harvester and harvested.

Perish defines the circumstances under which an animat instance may cease to exist during the simulation (e.g. "starve"). This class defines minimum thresholds related to animat internal state that determine conditions for the endurance of its existence. It may be associated with the animat class itself to set conditions by which an instance may cease to exist due to crowding. Associations with spatial environment variables may provide further conditions for existence.

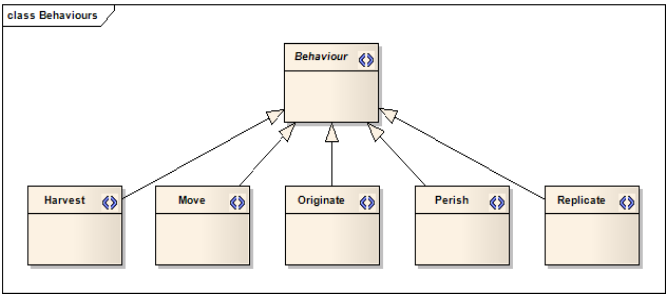

Figure 4: The five behavioural stereotypes.

This is just a selection of behaviour classes that provide a set of core procedures for animat conduct. In their seminal work, Epstein and Axtel [14] conceive a considerably larger set of behaviours in a spatially bound agent based simulation, including elaborate processes such as trade and cultural exchange. While interesting, these intricate behaviours are less common in pure GIS applications (where the dynamics is captured at a higher level of geographic abstraction) and more keen to

Social or Economics studies. However, the addition of further behaviours will be the main process of extension of DSL3S, answering to new requirements if necessary.

Views and Icons

DSL3S models can become visually intricate if a single diagram would de used to represent a complex spatial simulation scenario. To avoid such difficulties distinct Views are proposed to better organise models (see Figure 5):

- **Global View** - contains the simulation settings that do not have spatial realisation. Includes Simulation and Global classes, defining parameters such as the number of time steps to run, spatial extent or result output.
- **Spatial View** - where all the Spatial Environment variables are configured, defining the geographic inputs to the simulation.
- **Animat View** - defining animat internal state. In simulations scenarios with more than one animat several animat views can be used.
- **Behaviour View** - contains only the classes with specialisations of the Behaviour stereotype applied on, but also showing composition links to all classes that parametrise each behaviour. A dedicated view for each different behaviour is proposed.
- **Simulation View** – a simple package diagram that aggregates all other views.

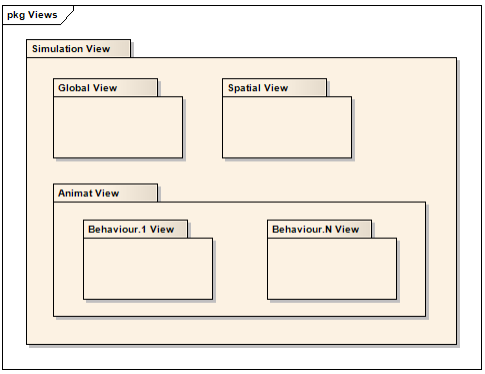

Figure 5: The proposed Views scheme for DSL3S.

Beyond these views a set of icons is also proposed to make the language visually explicit (see Figure 6). For Simulation, Global Variable and Spatial Environment Variable direct pictorial representations of their concepts are used. For the stereotypes Animat, State and Behaviours are proposed abstract symbols intending to create mental associations with a simulation model.

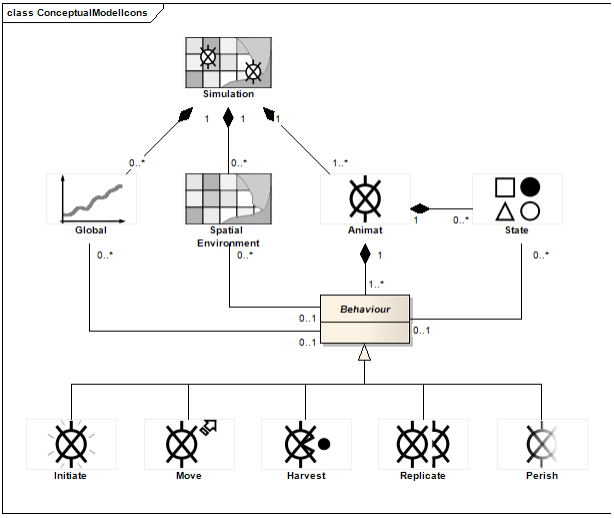

Figure 6: The proposed icons for DSL3S stereotypes.

Implementation

The reference architecture for the DSL3S language defines three components, all based on open source technologies (see Figure 7):

i. the UML profile, supported by the Papyrus add-on for the Eclipse[8] IDE;

ii. MASON, a Program-level tool supporting the generated code;

iii. model-to-code generation templates, developed with Acceleo, another Eclipse add-on.

Papyrus

Papyrus[9] was a project started by the *Commissariat à l'Énergie Atomique* in France, with the aim of producing an advanced graphical editor for the UML language, supporting particular DSL, especially SysML[10], a language for systems engineering. It evolved as an open source product based on the Eclipse Modelling Framework[11] (EMF), a tool-kit that supports the edition and visualisation of structured models defined in the XMI language and provides a set of Java classes that facilitate its

8 http://www.eclipse.org

9 http://www.eclipse.org/modeling/mdt/papyrus/

10 http://www.sysml.org/

11 http://www.eclipse.org/modeling/emf/

manipulation. Papyrus eventually evolved to support the development of *ad hoc* DSL, through the definition of UML profiles.

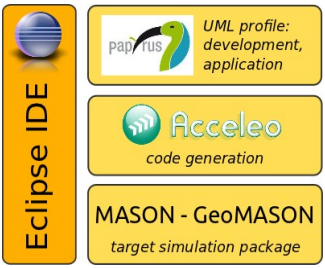

Figure 7: The technologies used to implement DSL3S.

In 2009 Papyrus merged with Eclipse altogether, becoming the forefront graphical UML editor add-on for this popular IDE, replacing other assorted tools with fewer capabilities. It is presently close to fully support the version 2 of the UML language, coming to be one of the most advanced tools available for the purpose. Beyond SysML, Papyrus includes a series of other DSL dedicated to domains like embedded systems or automotive systems.

Acceleo

Acceleo[12] is an open source code generator created by the French company Obeo, first released in 2006, as a plug-in to Eclipse 3.0 and 3.1. It is also built on EMF, facilitating the interoperability with several other modelling tools based on the same technology. The following year the Eclipse Foundation took Acceleo as an official project. In latter versions Acceleo adopted the MOF Model to Text Transformation Language[13] (MOFM2T), another OMG standard. Though not yet fully implementing this standard, the model-to-code generators produced with Acceleo are today some of the closer to the scheme proposed by the OMG.

The code generation mechanism is based on special files called templates, which define the text output to produce from a graphical model. They are composed by regular text plus a series of annotations that are substituted by values and names of model elements during generation time (see Table 1). Traditional computational operations such as branches or loops are also possible to include with specific annotations, allowing the production of more complex outputs. Templates can be articulated through an inclusion mechanism, whereby a master template can make use of several other templates creating a generation chain. When fully developed, a generation chain can be transformed into an independent plug-in for Eclipse, facilitating its portability and application.

12 http://www.acceleo.org/pages/introduction/en

13 http://www.omg.org/spec/MOFM2T/1.0/

Acceleo 3 fully supports code generation from meta-models, identifying stereotypes applied on classes and providing access to properties. The later isn't based on MOFM2T, but provided by a service, essentially a Java method that browses through the UML2 object model associated with each class.

Acceleo code template	Sample output

`[comment encoding = UTF-8 /]`
`[module generateProf(`
`'http://www.eclipse.org/uml2/3.0.0/UML')/]`
`[template public generateProf(c : Class) ?`
`(c.hasStereotype('Table'))]`
`[comment @main /]`
` [if (c.hasStereotype('Table'))]`
` [file (c.name.concat('.test'), false, 'UTF-8')]`
`  File for a class with the stereotype Table.`
`  The name of the class is: [c.name/].`
` [/file]`
` [/if]`
`[/template]`
	`File for a class with the stereotype Table.`
	
	`The name of this class is Clients.`

Table 1. A simple Acceleo template and its output when used on a class named “Clients” with the stereotype “Table” applied on. The “hasSteretype” query is an external service.

MASON

MASON (the acronym for “Multi-Agent Simulator Of Neighbourhoods”) aims to be a light-weight, highly-portable, multi-purpose agent-based modelling package [15]. MASON is a relatively new tool, with the first version coming to light in 2003; being somewhat different for tools developed in the 1990s. Its objects are architected in such a way that simulation models are totally isolated from visualisation and input/output mechanisms. MASON is fully written in Java and freely distributed, hence it produces programs that are highly portable between different operating systems, not only running alike but also presenting identical results. Comparative results have shown that in general MASON is likely the fastest of the main Program-level tools.

MASON was initially used for artificial scenarios, evolving as a text input/output tool, lacking graphical facilities [7]. These features are now fully available, and improved with the usage of Java3D. Supported by extensive documentation and a relevant comunity[14], MASON has been adopted more widely.

GeoMason[15] is a rather complete extension for geo-referenced data. Input and output functionality is available for both raster and vector data, relying on third party packages: the Java Topology Suite for geometry manipulation, GeoTools for vector interaction and GDAL for raster formats.

Its light-weight infrastructure, extensive documentation, and easy of integration through Eclipse made MASON an obvious choice for validating DSL3S.

14 http://cs.gmu.edu/~eclab/projects/mason

15 http://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/

Summary and future work

The application of spatial simulation techniques to the GIS realm is still today locked in the choice between versatile tools that require advanced programming skills and easy to use pre-built models that force relevant compromises of transparency and scope. Several DSL, such as NetLogo or MOBIDYC, have been tried in this field but invariably producing imperative languages with compromises of their own.

DSL3S proposes a new approach to this subject, with the development of a UML profile language and a code-to-model transformation infrastructure, producing simulation models based on a Program-level tool. This scheme promotes faster model development, reduces coding errors and increases model readability through graphical models. Relying on MASON, it guarantees interoperability with geographic data, while largely dispensing coding activities. The language is being developed on the Eclipse IDE, using the modelling ad-ons Papyrus and Acceleo.

In the near future DSL3S will be assessed through its application to real world scenarios. This interative process will allow to understand how far it can go in its current form and how necessary new behavioural stereotypes may be.

References

[1] Wuensche , A. and M. Lesser (1992). “The Global Dynamics of Cellular Automata”. Addison-Wesley.

[2] J. Ferber (1999). “An Introduction to Distributed Artificial Intelligence”, chapter Multi-Agents Systems, Addison-Wesley.

[3] Fall, A. and Fall, J. (2001), "A domain-specific language for models of landscape dynamics", Ecological Modelling, 141, pp. 1–18.

[4] Olde, V. and Wassen, M. (1997), "A comparison of six models predicting vegetation response to hydrological habitat change", Ecological Modelling, 101, pp. 347–361.

[5] de Sousa, L. and da Silva, A. (2011), “Review of Spatial Simulation Tools for Geographic Information Systems” in SIMUL 2011, The Third International Conference on Advances in System Simulation, ThinkMind.

[6] N. Collier, T. Howe, and M. North (2003). “Onward and upward: the transition to Repast 2.0”, In First Annual North American Association for Computational Social and Organizational Science Conference, Pittsburgh, Pa.

[7] Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. (2005) “Mason: A multi-agent simulation environment”. Simulation: Transactions of the society for Modeling and Simulation International, 82(7), pp. 517–527.

[8] Tobias, R. and C. Hofmann. (2004) Evaluation of free java-libraries for social-scientific agent based simulation. Journal of Artificial Societies and Social Simulation, 7(1).

[9] Samuelson, D. and C. Macal. (2006) “Agent-based simulation comes of age”. OR/MS Today, 33(4), pp. 34–38.

[10] Mladenoff, D. (2004) “Landis and forest landscape models”. Ecological Modelling, 180(1), pp. 7 – 19.

[11] Merzenich, J. and L. Frid. (2005) “Projecting landscape conditions in southern Utah using VDDT”. In M. Bevers and T. M. Barrett, editors, Systems Analysis in Forest

Resources: Proceedings of the 2003 Symposium, pp. 157–163, Portland, OR. U.S. Department of Agriculture, Forest Service, Pacific Northwest Research Station.
[12] Ginot, V., C. Le Page, and S. Souissi. (2002) "A multi-agents architecture to enhance end-user individual based modelling". Ecological Modelling, 157, pp. 23– 41.
[13] S. W. Wilson (1991), "The animat path to AI". In J. A. Meyer and S. Wilson, editors, From Animals to Animats, pp. 15–21, Cambridge, MA. MIT Press.
[14] Epstein, J. M., and R. Axtell. (1996) "Growing Artificial Societies". MIT Press.
[15] S. F. Railsback, L. L. Steven, and J. K. Jackson (2006), "Agent-based simulation platforms: Review and development recommendations". Simulation, 82, Issue 9.