GAMAGRAM: graphical modeling with the GAMA platform

Patrick Taillandier

Proceedings of ICCSA 2014
Normandie University, Le Havre, France - June 23-26, 2014

GAMAGRAM: GRAPHICAL MODELING WITH THE GAMA PLATFORM

Patrick Taillandier *†

Abstract. These last years have seen the multiplication of platforms dedicated to the conception and simulation of agent-based models for studying complex systems. If these platforms allowed to democratize this modeling approach, they are often complex to use by non-computer scientists as most of them require to define models by writing numerous code lines. In this paper, we present GAMAGraM, a new graphical modeling plug-in integrated into the GAMA platform. This plug-in allows users to define models from a graphical interface. Defining a model with the GAMAGraM plugin means first defining the structure of the model by a conceptual entity-relationship diagram, then defining the properties of entities through dedicated dialog boxes. The graphical editor provides autocompilation tools allowing to ease the work of the modelers.
Keywords. Complex System, Agent-based Modeling, Graphical Modeling, GAMA Platform

1 Introduction

Nowadays, agent-based modeling is more and more used to study complex systems, in particular by researchers coming from other fields than computer sciences (geography, environmental sciences). Unfortunately, the use of such modeling approach requires to have skill in programming, in particular when the system to model integrates social and spatial aspects. This issue has for consequence that most of the models are still developed by computer scientists and not directly by domain experts, which slows the diffusion of this modeling approach and the transfer of knowledge to the decision-makers.

In order to tackle this issue, some agent-based modeling platforms have proposed to let the user define his/her models thanks to a graphical interface allowing to minimize the quantity of lines of code to write. However, these platforms are often complex to use or very limited in terms of models they allow to build.

The GAMAGraM plugin has for ambition to be simple to use and at the same time very powerful. It is based on the GAMA platform [12] that allows to define large-scale models integrating rich spatial and social dynamics (e.g. MAELIA [19, 11], MIRO [4]).

*Patrick Taillandier is with UMR CNRS IDEES, University of Rouen, France. E-mail: patrick.taillandier@univ-rouen.fr
†Manuscript received April 19, 2009; revised January 11, 2010.

The paper is organized as follows: Section 2 presents the context of this work, in particular the existing agent-based modeling platforms and the graphical modeling languages. Section 3 is dedicated to the presentation of the GAMAGraM plugin. At last, Section 4 concludes and presents perspectives.

2 Context

2.1 existing platforms

Nowadays, there are numerous platforms dedicated to the agent-based modeling of complex systems. These platforms can be divided in 3 - non-exclusive - groups according to the type of programing language used to define models.

The first group is composed of the platforms that require to define models through a high-level generic programming language (Java, C++, Python...). These platforms are most of the time intended to computer scientists and are the most adapted to the development of large-scale models. Repast Symphony [17], MASON [15] and SWARM [16] belong to this category.

The second group is composed of the platforms that provide a dedicated modeling language. These platforms are most of the time easier to use than the ones of the first group. There are intended to a wider range of users. However, they require some skills in algorithmic. Netlogo [20] and GAMA [12] belong to this group.

The last group is composed of the platforms that allow to define a model through a graphical modeling language. These platforms require less skills in computer science than the platforms belonging to the two other groups. Moreover, they offer the advantage to ease the discussion between modelers and domain-experts/stakeholders and are thus particularly adapted to be used in a participatory modeling context.

StarLogo [18] and Modelling4All [13] belong to this group. These tools that can be used by all types of users are mostly pedagogical tools and are limited to the development of simple models.

Another platform that belongs to this group is Repast Symphony. It proposes to define models through three ways: by using Java, using the ReLogo language or

through a graphical modeling language [17]. If for simple models (or rapid prototyping) the ReLogo language and the graphical modeling tools can be used, developing a complex model with this platform requires to have knowledge in Java.

In its last version, the Cormas platform [8] provides as well some graphical modeling tools (definition of activity diagram). However, this platform, specialized for participatory modeling, does not offer the same richness as GAMA or Repast Symphony in terms of model development, in particular for the development of models based on vector geographical - GIS - data.

At last, the MAGeo platform [14] allows to simply define a model through a dedicated graphic interface. It proposes to formalize the agent behavior as an aggregation of basic behaviors with a simple grammar. This grammar is perfectly adapted to the definition of simple agents, but does not allow (or not directly) to define more complex agents.

To conclude on the existing platforms allowing to develop models through a graphical interface, they are either too complex to use for non-computer scientists (e.g. Repast Symphony) or too limited to develop large-scale models (e.g. StarLogo, Modelling4All, Cormas, MAGeo).

2.2 graphical modeling

Numerous graphical languages were proposed for modeling purpose. The most famous and used one is UML. In the context of agent-based modeling, some works have shown the interest of using such graphical language for communication [6]. However, some authors have point out that the use of UML as an agent-oriented modeling language is inappropriate [7].

Other graphical language based on UML and dedicated to multi-agent systems have been proposed: the most famous ones are AUML [5] and AML [9]. These languages allow to introduce some specific features linked to the agent paradigm. However, their scope goes beyond the agent-based modeling and covers all the multi-agent aspects, which can make these languages difficult to apprehend by non-computer scientists.

A last modeling language to cite is the one proposed by the MAGeo platform. This language is based on the AOC (Actor - Organization - Behavior) meta-model [10]. This graphical language is close to the UML one and respects most of the properties of the OOP (Objected-Oriented Programming). In addition, it allows to natively define multi-level models. However, this language does not allows to define complex behaviors for the agents. In particular, no difference is made between what an agent can do and what it is going to do (capabilities versus behavior).

For the GAMAGraM plugin, our goal is to propose a modeling language simple to manipulate (with a very small number of concepts) and that allows to develop large-scale models. In order to achieve this objective, we identified several properties that our modeling language should respect:

- Properties of the OOP
- Differentiate what an agent can do and what it is going to do (capabilities versus behavior)
- Native handling of multi-level modeling
- Possibility to define elements related to the simulation visualization

To conclude, if numerous graphical modeling languages exist, none of them covers all the properties that we have identified. Moreover, generic languages such as UML, A-UML and AML can be more complex to use as they propose many features that are not useful for agent-based modeling and lack of specific features that can help modelers.

3 Modeling with GAMAGraM

3.1 General properties of GAMAGraM

GAMAGraM has for objective to fill the need in platforms accessible to the highest number and at the same time allowing the definition of large-scale models. We chose to develop GAMAGraM as a plugin of GAMA, because this open-source platform provides already numerous features to develop models, in particular concerning the management of GIS data. Moreover, GAMA is easily extensible.

GAMAGraM allows GAMA users to graphically define their models and eventually to translate them to the GAML language (GAMA Modeling Language). In addition, GAMAGraM allows to translate a GAML model into a graphical one. This feature aims at facilitating the discussions (and communication) about a model. GAMAGraM is based on the Graphiti plugin of Eclipse [3].

3.2 Definition of the model structure

The modeling process with GAMAGraM consists first in defining a conceptual model consisting in a entity-relationship diagram, then to fill all the defined entities through dialog boxes.

We chose to base the conception of the conceptual model on a new modeling language based on the GAMA meta-model. Indeed, if many agent-oriented meta-models were proposed in the literature (see [7] for a presentation of the most famous ones), most of them are not directly dedicated to simulation purpose and very difficult to grasp for non-computer scientists. Another advantage of using the GAMA meta-model is to limit the gap between the conceptual model and the final implemented GAMA model.

Figure 1 presents the meta-model of GAMA. The main component of this meta-model is the **Species**. A **Species**, like a class in OOP, defines the common characteristics to all the agents of a population. In particular,

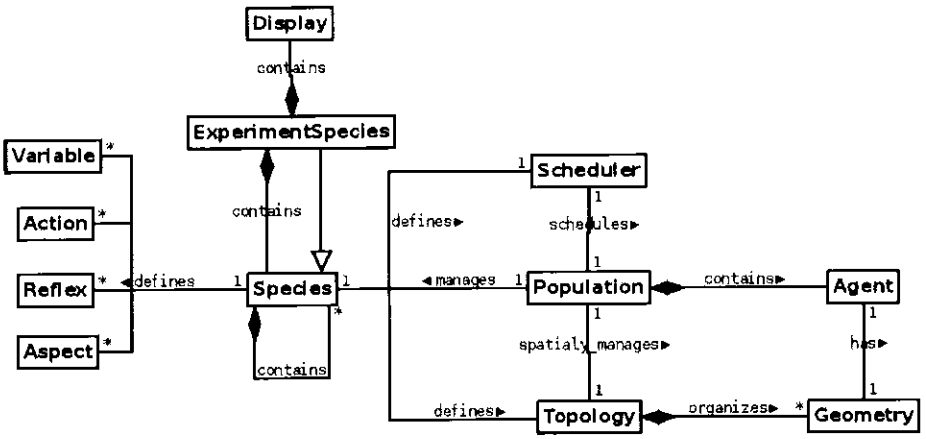


Figure 1: Meta-model of GAMA

it defines their variables, their actions, their reflexes and their aspects. An **Action** is a capabilities that the agents of the population have, i.e. something that the agents can do. A **Reflex** is a behavior, i.e something that the agents of the population are going to do (if some conditions are respected). An **Aspect** represents a possible display of the agents. Note that a species can specifies several actions, reflexes and aspects. In addition, a species specifies the spatial topology and scheduling of the agent population. A containment relationship between species allows to describe the hierarchical levels of agency. At last, a specialization relationship between species allows to define inheriting links between them. An **ExperimentSpecies** represents a context of execution of a model. It is a particular species that contains a set of species (the one defined in the model) and a set of displays.

More details about the meta-model of GAMA can be found in [21]. The use of this meta-model allows to respect the 4 properties defined in Section 3. Note that this meta-model is close to the AOC one, but offers more freedom in terms of agent behavior definition.

Figure 2 presents the modeling graphical framework of GAMAGraM. The right palette allows to select the type of elements to add to the diagram. This framework proposes all the classic features of graphical editors (undo, drag and drop....).

Table 1 presents all the elements that can be added to the conceptual diagram.

When a graphical model is created, a first species of agents is automatically created: the world species. The world species corresponds to the first level of agency that describes the global spatial topology of the model, its basic scheduling, its parameters and global behaviors, and is the host of the populations of agents described by the species written by the modeler.

Thus, the development of the conceptual model consists in defining all the species (with their chosen topology: continuous, grid) living in the world, their capabilities (actions), their behavior (reflexes) and possible displays (aspects). Note that the inheriting relation can be used between species. In addition, the definition of the conceptual model consists in defining the possible contexts of execution of the simulation (experiments) and for each of them the corresponding outputs (displays). Each time the user modifies the diagram, this one is validated: if there is no error in the diagram, all its components appear with green borders, and buttons corresponding to each defined experiments appear in the top of the editor (for example, see the *my_GUI_xp* button in Figure 2). But clicking on one of the experiment buttons, the user can load it (and run the corresponding simulation(s)). If there are errors in the diagram, the problematic components appears with red borders.

As an example, Figure 3 presents the conceptual model of a simple predator prey model. In this model, 4 species of agents live in the world:

- *vegetation_cell*: species with a grid topology that will be used as spatial environment for the other agents. This species has only one behavior (reflex): *grows*.
- *animal*: species with a continuous topology that will be used as the generic species to define the predators and preys. This species has 4 behaviors (reflex): *eats*, *moves*, *reproduces* and *dies*. It also has one action called *eating* and one aspect called *circle*.
- *prey*: species with a continuous topology that inherits from the *animal* species. This species overrides the *eating* action.
- *predator*: species with a continuous topology that inherits from the *animal* species. This species overrides the *eating* action.

In addition, we define one GUI experiment called *main_xp* that has a display called *map* and a display called *charts*. A tutorial describing how to build this diagram can be found on the GAMA website [2].

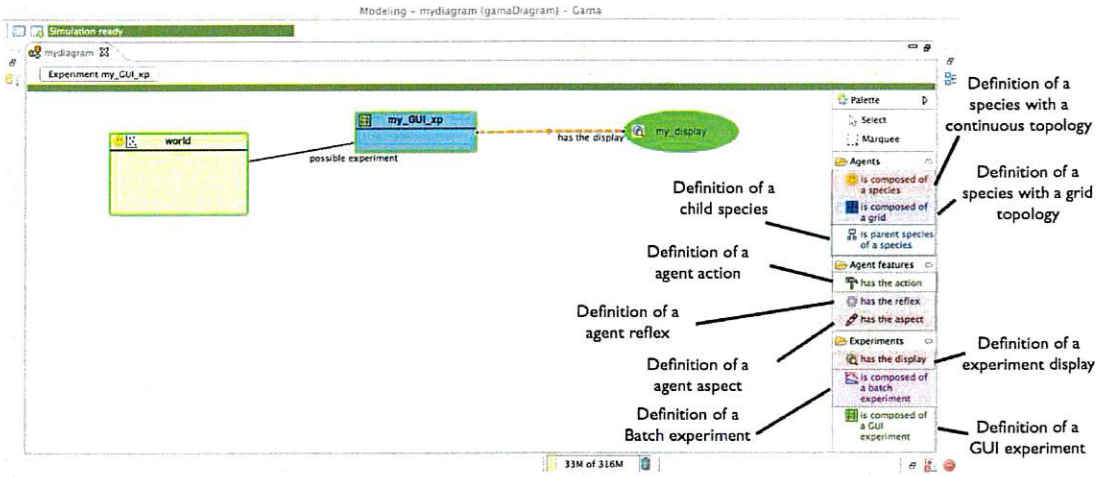
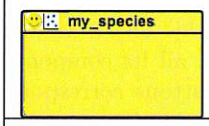
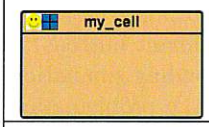
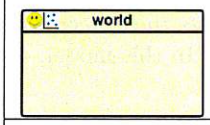
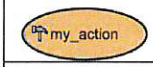
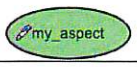
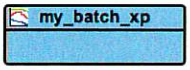


Figure 2: Graphical User Interface of the GAMAGraM plug-in

Table 1: Entities of the graphical modeling language

Symbol	Source	Description
	A species	**Species**: A species of agents with a continuous topology.
	A species	**Grid**: A species of agents with a grid topology
	-	**World**: the first level of agency. It contains all other species of agents.
	A species	**Action**: A capability that the agents have.
	A species	**Reflex**: A behavior that will be activated at each simulation step (according to a given condition).
	A species	**Aspect**: A possible display for the agents.
	The world	**GUI Experiment**: load only one simulation with the graphical user interface
	The world	**Batch Experiment**: load a set of simulations without the graphical user interface
	A GUI Experiment	**Display**: frame allowing to display outputs (map, charts...)

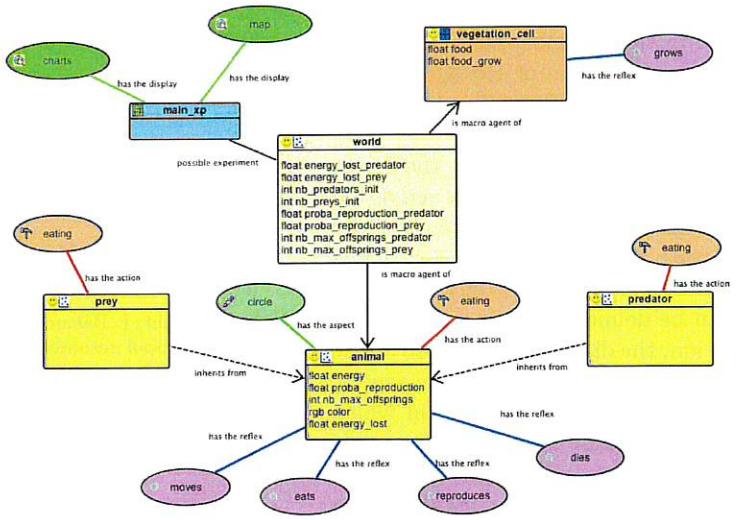


Figure 3: Conceptual model of the predator-prey model

3.3 Definition of the properties of each entity

Once the conceptual model defined, the next step consists in describing the properties of each defined entities.

When the user clicks on an entity, a new dialog box allowing to parameterize it appears. It is through these dialog boxes that the modeler will be able to transform his/her conceptual model into a simulation. Most of the time, the parameterization will just consist in making a choice between different options, but sometimes it will consist in writing GAML instructions. Note that a complete description of the GAML language can be found in the GAMA documentation [1].

The most important entity to parameterize is the species one. The species dialog box allows to define many properties of the species without having to write code (see Figure 4). In particular, it allows to define the geometry of the agents (point, line, or polygons) and variables. For each variable the modeler has to define its name and its type (among many types such as integer, float number, string, list, matrix, map, point, geometry, graph, path...). In addition, the modeler can define optional facets for each variable such as its initial value, a minimum, a maximum, an expression that will be used to re-compute the variable at each simulation step or a function that defined how the variable will be computed each time it is called. In addition, the modeler can give skills to the species. A skill is a predefined set of variables and actions coded in Java. For instance, the moving skill gives to the species the variables *speed*, *heading* and *destination* and the actions *move*,*goto*,*wander* and *follow*. At last, the modeler can define an *init* block that represents the constructor of the species, i.e. define what will happen at the creation of the agents.

Note that the dialog boxes for the world species and the grid definition are very similar. For the grid, the

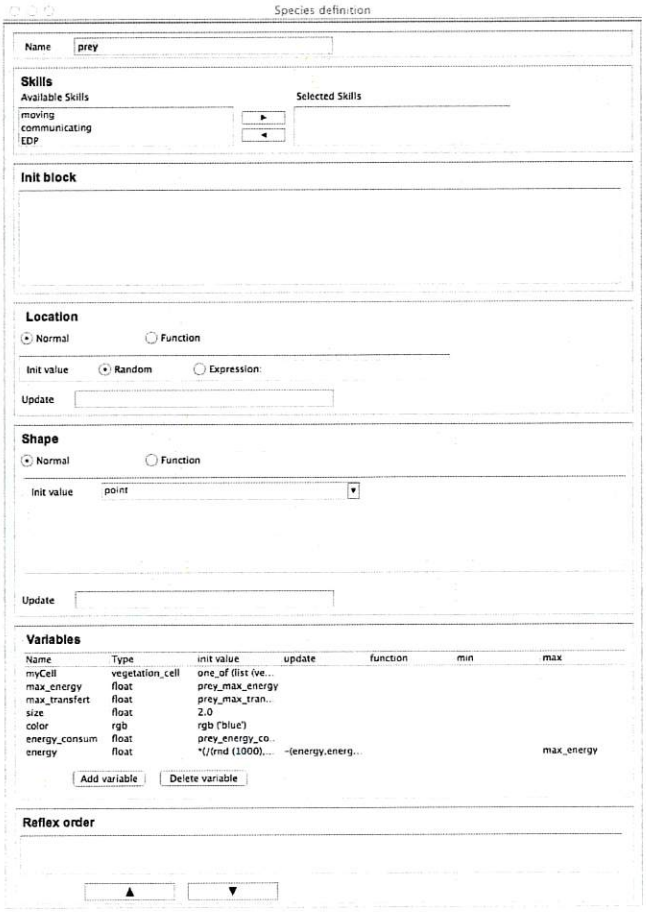


Figure 4: Dialog box for Species definition

dialog box allows just to additionally define the number of cells in the rows and in the columns, and the type of neighborhood: Moore, van Neumann or Hexagonal. For the world species , the dialog box allows to additionally define if the environment is torus or not.

Concerning the parameterization of reflexes, the dialog box allows to choose the condition of the reflex activation and its effect. The activation condition and the effect are described using the GAML language. The action dialog box is very similar to the reflex one. The only difference is that no condition can be defined.

For the aspect definition, the dialog box allows to define the layers (and their order) that will compose the aspect. These layers are defined through a dialog box, in which the modeler can choose the shape to display (a simple shape such as a circle, a square, a rectangle..., an icon, a text or a complex geometry such as a polyline or a polygon), its color, and some specific properties (rotation, fill/empty shape...).

The experiment definition dialog box allows to define the parameters that the user will be able to modify through the simulation interface.

At last, concerning the display definition, the dialog box allows to define the layers that will compose the display (and their order), to choose a color for the background and the refreshing rate. The layers are defined through a dialog box, in which the modeler can choose the elements to display (a list of agents, a chart, an image, a text...), the level of transparency of the layer, its size and its position.

The complete description of these dialog boxes can be found on the game website [2].

4 Conclusion

In this paper, we presented the GAMAGraM plugin that allows to define agent-based models through a graphical interface. This plug-in, which is under GPL license, is available from the GAMA website [2].

In order to validate our claims concerning the ease of use of our plugin and its powerfulness in comparison to others platforms, we plan to carry out tests with end-users (in particular geographers).

In addition, we plan numerous improvements for the plugin. A first one will consist in adding tools to define the reflexes and actions of agents directly through a graphical interface. We plan for that to add the possibility to define these elements thanks to an activity diagram. At last, we plan to add a WYSIWYG interface for the display definition.

References

[1] Gama modeling guide 1.6: https://code.google.com/p/gama-platform/wiki/modelingguide16, January 2014.

[2] Gama website: https://code.google.com/p/gama-platform, January 2014.

[3] Graphiti: http://www.eclipse.org/graphiti/, 2013 October.

[4] A. Banos, N. Marilleau, and M. Team. Improving individual accessibility to the city: an agent-based modelling approach. In *ECSS*, 2012.

[5] B. Bauer, J. Müller, J. Odell, and A. Arbor. Agent uml: A formalism for specifying multiagent interaction. *Agentoriented software engineering*, 1957:91–103, 2001.

[6] H. Bersini. Uml for abm. *JASSS*, 15(1):9, 2012.

[7] G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J. J. Gomez-Sanz, J. Pavon, and C. Gonzalez-Perez. Faml: a generic metamodel for mas development. *Software Engineering, IEEE Transactions on*, 35(6):841–863, 2009.

[8] F. Bousquet, I. Bakam, H. Proton, and C. Le Page. Cormas: Common-pool resources and multi-agent systems. In *Tasks and Methods in Applied Artificial Intelligence*, pages 826–837. Springer Berlin Heidelberg, 1998.

[9] R. Cervenka, I. Trencansky, and C. M. Modeling social aspects of multiagent systems: the aml approach. In *AOSE*, 2005.

[10] E. Daudé, P. Langlois, B. Blanpain, E. Sapin, et al. Aoc, une ontologie formelle pour la modélisation de systèmes complexes en géographie. In *Outils, méthodes et modèles en géomatique pour la production de connaissances sur les territoires et le paysage*, 2010.

[11] B. Gaudou, C. Sibertin-Blanc, O. Therond, F. Amblard, J. Arcangeli, M. Balestrat, M. Charron-Moirez, E. Gondet, Y. Hong, T. Louail, E. Mayor, D. Panzoli, S. Sauvage, J. Sanchez-Perez, P. Taillandier, V. Nguyen, M. Vavasseur, and P. Mazzega. The maelia multi-agent platform for integrated assessment of low-water management issues. In *MABS*, 2013.

[12] A. Grignard, P. Taillandier, B. Gaudou, D. Vo, N. Huynh, and A. Drogoul. Gama 1.6: Advancing the art of complex agent-based modeling and simulation. In *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, volume 8291 of *Lecture Notes in Computer Science*, pages 117–131, 2013.

[13] K. Kahn and H. Noble. The modelling4all project a web- based modelling tool embedded in web 2.0. In *International Conference on Simulation Tools and Techniques*, 2009.

[14] P. Langlois, B. Blanpain, and E. Daudé. Magéo, une plateforme de simulation multi-agents pour tous. In *SimTools*, 2013.

[15] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan. Mason: A new multi-agent simulation toolkit. In *SwarmFest Workshop*, volume 8, 2004.

[16] N. Minar, R. B. Y, and C. L. Z. The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, Santa Fe Institute, 1996.

[17] M. North, N. Collier, J. Ozik, E. Tatara, C. Macal, M. Bragen, and P. Sydelko. Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):3, 2013.

[18] M. Resnick. Starlogo: an environment for decentralized modeling and decentralized thinking. In *Conference companion on Human factors in computing systems*, pages 11–12, 1996.

[19] P. Taillandier, O. Therond, B. Gaudou, et al. A new bdi agent architecture based on the belief theory. application to the modelling of cropping plan decision-making. In *IEMSs*, 2012.

[20] S. Tisue and U. Wilensky. Netlogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, pages 16–21, 2004.

[21] D.-A. Vo, A. Drogoul, and J. Zucker. An operational metamodel for handling multiple scales in agent-based simulations. In *RIVF*, pages 1–6. IEEE, 2012.