

RatKit: Repeatable Automated Testing Toolkit for Agent-Based Modeling and Simulation

İbrahim Çakırlar¹(✉), Önder Gürcan^{1,2}, Oğuz Dikenelli¹,
and Şebnem Bora¹

¹ Department of Computer Engineering, Ege University,
35100 İzmir, Turkey

icakirlar@gmail.com, {onder.gurcan,
oguz.dikenelli, sebnem.bora}@ege.edu.tr

² LIST, Laboratory of Model Driven Engineering for Embedded Systems, CEA,
Point Courrier 174, 91191 Gif-sur-Yvette, France
onder.gurcan@cea.fr

Abstract. Agent-based modeling and simulation (ABMS) became an attractive and efficient way to model large-scale complex systems. The use of models always raises the question whether the model is correctly encoded (verification) and accurately represents the real system (validation). However, achieving a sufficiently credible agent-based simulation (ABS) model is still difficult due to weak verification, validation and testing (VV&T) techniques. Moreover, there is no comprehensive and integrated toolkit for VV&T of ABS models that demonstrates that inaccuracies exist and/or which reveals the existing errors in the model. Based on this observation, we designed and developed RatKit: a toolkit for ABS models to conduct VV&T. RatKit facilitates the VV&T process of ABMS by providing an integrated environment that allows repeatable and automated execution of ABS tests. This paper presents RatKit in detail and demonstrates its effectiveness by showing its applicability on a simple well-known case study: predator - prey.

Keywords: Agent-based modeling and simulation · Model testing · Verification and validation

1 Introduction

Agent-based modeling and Simulation (ABMS) is a very multidisciplinary complex system modeling and simulation technique, which has been used increasingly during the last decade. The multidisciplinary scope of ABMS ranges from the life sciences (e.g. Biological Networks [6], Ecology [7], social Sciences [8], Scientometrics [9] to Large-scale Complex Adaptive CommunicatiOn Networks and environmentS (CACOONS) [10] such as Wireless Sensor Networks and the Internet of Things (IoT)). While in some domains, ABMS is used for understanding complex phenomena, in other domains it is used for designing complex systems. However, whatever the objective is, in all of these domains large sets of agents interacting locally give rise to bottom-up collective behaviors. The collective behaviors of agents, whether emergent or not [11], depend on the local competences, the local perceptions and the partial

knowledge of agents as well as the global parameter values of the simulation run. A slight difference in any of these properties (whether intentional or not) may result in totally different collective behaviors. Such a consequence leads either to a misunderstanding of the system of interest or a bad system design.

Besides, despite all ABMS platforms are developed by computer scientists, the users of these platforms (i.e. The developers of ABMS models) are more heterogeneous. Depending on the application domains, they can be (1) computer scientists that are building ABS models for their domains, (2) non-computer scientists that are building models for their domains or (3) computer scientists that are working closely with non-computer scientists. On the one hand, non-computer scientist modelers are experts in their domains (i.e. Domain experts) and are said to be capable of building the right models. However, translating these models into their corresponding software models (i.e. ABS models) can sometimes be problematic and open to mistakes. Moreover, since they have less expertise concerning software development, it is a big mystery as to whether they are building the models right or not. On the other hand, computer scientist modelers are better at building models correctly, but they usually lack the expertise to build the right models.

In this sense, correct design and implementation of ABS simulation models are becoming highly important to increase reliability and to improve confidence. The use of models always raises the question whether the model is correctly encoded (verification) and accurately represents the real system (validation). Model verification deals with “*building the model right*” while model validation deals with “*building the right model*”, as stated in [1]. Model verification is the process of determining that a model is meeting specified model requirements and reflecting the system of interest accurately. Also, model validation is the process of determining the degree to which a model is an accurate representation of the system of interest from the perspective of the intended modeling objectives. Both verification and validation are processes that gather evidence of a model’s reliability or accuracy; thus, verification and validation (V&V) cannot prove that a model is definitely correct and accurate for all possible scenarios, but, rather, it can provide evidence that the model is sufficiently accurate for its intended use [27].

In the literature, there are two main focuses for building accurate ABS models: model fitting and model testing [28]. For both of these focuses the ultimate goal is to have an ABS model that appropriately mimics the real system, however each focus has different ways to achieve this. Model testing demonstrates that inaccuracies exist in the model and reveals the existing errors in the model. In model testing, test data or test cases are subject to the model to see if it functions properly [2]. Model testing focuses on observable behaviors of the real system according to the experimental or real data. On the other hand, model fitting focuses on achieving non-observable behaviors of the real system that are not gathered with scientific or experimental methods. Even though these focuses are at opposite ends of the spectrum, and certainly hybrids of these focuses offers the solution of aforementioned V&V requirements of ABS models.

Model testing and model fitting, are general techniques that can be conducted to perform verification and/or validation of ABS models. In this study, we extend the model testing scope with the requirements of model fitting. We aimed to enrich model testing methods with test scenario visualization, visual tests and logging support in order to support model fitting, especially for domain experts and non-computer

scientists. As [12] points out traditional testing techniques for VV&T cannot be transferred easily to ABS. There are some efforts [13–17], but these studies do not directly deal with model testing processes and focus on late validation and verification. As well, there are few proposed model testing frameworks to conduct validation and verification throughout the model testing process [15, 18, 19]. Among them, [18] proposed an integrated testing framework, but unfortunately this framework not easy to use for non-computer scientists.

Based on the above observations, our desire is to develop an automated and integrated testing framework for ABSs in order to facilitate the model testing process for all types of model developers. Towards this objective, we took the generic testing framework proposed by Gürçan et al. [18] and improved it one step further by taking into account the requirements of testing frameworks for ABMS. Previously, testing requirements for ABMS are defined and testing levels of ABMS that can be subject of the model testing process are clarified in our previous studies [18, 22]. We also revise our multi-level testing categorization by keeping in sight the requirements of ABS testing frameworks.

2 Requirements of Agent Based Simulation Testing Frameworks

VV&T leads the simulation model development to increase understanding of the potential of models and to decide when to believe a model, and when not to, and to interpret and to use the model’s results [29]. However, it should be noted that VV&T is not a silver bullet. VV&T also has some limitations and constraints. Apparently, one intending to design a testing framework should take into consideration the requirements below.

- **Integrity:** Testing of the model is not separate from the model development (especially since it covers the verification). Rather, these tasks are tightly coupled, since a testing framework for ABMS should be integrated or pluggable to the simulation environment in order to behave like a simulation engine, to interpret the model outputs and to execute the testing criteria corresponding to the evaluation rules. Thus, applying VV&T in the early steps in model development can be easily achieved.
- **Multilevel Testing:** Multilevel testing involves testing the model elements at different levels of organization; micro-, meso- and macro-levels. Due to the multilevel nature of ABMS [25] and experiences reported in the literature [26], obviously a testing framework dedicated to ABMS should support multilevel testing as discussed in other studies [18, 22, 24]. Corresponding to the multilevel testing, model components tested at lower levels can be used in upper levels. So, multilevel testing does not distinguish testing requirements. Rather, it presents a systematic way to test simulation models iteratively.
- **Automated Testing:** Automatic testing is the capability of executing model tests together and individually. Multilevel tests are not independent from each other. Moreover, each level is a prerequisite of the upper-level for proper testing. Testing

model with the model development in a multilevel manner systematically organizes the model to achieve intended reliability and accuracy. Therefore, automated testing is the essential requirement for the complementarity of the testing levels. Thus, the impact of the new model components or behaviors added to the model can be easily understood without any extra effort.

- **Monitoring:** Monitoring the simulation models, the behaviors of agents, or occurrence of special or unexpected cases are the main expectations for testing. Monitoring an agent in micro-level, a group of agent in meso-level or the whole model in macro-level testing is the main requirement to evaluate models. Such a testing framework should provide evidence to the modelers in order to assess about the model behaviors or outputs. However, monitoring should be conducted without any intervention to the model behavior if we want reliable information about the model. Most of the existing monitoring efforts [15, 18] prefer to intervene in the scheduling of agents, agent behaviors or the simulated environment. In this case, observations gathered may be different from the real outputs of the model.
- **Parameter Tuning:** Simulation parameters are the key values for the model and affect the simulation behaviors. A dedicated testing framework should provide parameter tuning capability [5] to the modeler to find appropriate parameter values, showing the domino effect between parameters, testing the variety of parameter values, drawing the boundaries for the parameter value set, testing the parameter sensitivity, etc. To perform parameter tuning modelers choose to run the model multiple times with different initial values in order to find the appropriate values. But instead of such a testing framework supports parametric test scenario definition can handle this issue.
- **Presenting Model Observation:** Evaluation of the model observations against the real data is the subject of testing [2]. Model observations are not only final outputs, but also the data that are captured at any time during the model execution. An observation value can be the value of an agent attribute, state, parameter, environment parameter or resource. Presenting observations to the modeler contributes to evaluate the potential of the model.
- **Visualization Support:** VV&T of ABMS do not only focus on quantitative methods [15]. Especially for non-computer scientists, a testing framework should present visual outputs to support model fitting[. However, visualization is not only to visualize the simulation execution, but also to present or to summarize observations. Drawing a graphical representation of observation history should help modelers to review simulation execution or the behaviors of the agents. In this sense, modelers can monitor the behaviors of agents or a group of agents with different conditions without any extra effort. To perform VV&T based on classical quantitative techniques narrows the VV&T perspective. However, a testing framework that is enriched with the support of visualization provides a broader perspective in order to evaluate the potential of the model.
- **Logging:** Logging [15] is presenting a history of the model execution to the modeler. Some of the situations not considered in a test scenario can be determined with the help of logs. Especially in meso- and macro- testing levels, when the number of agents in the model under test increases, the impact of logging during assessment can be easily achieved. Reviewing logs help modelers to monitor the

model behaviors easily. Logging should be optional and should support logging levels in order to avoid confusion.

- **Ease-of-Use:** Testing proposals [13–17] for ABS is hard-to-use and requires extra effort. VV&T is difficult enough for modelers because of its nature. Therefore, it should be identical to the model development to address all modelers, especially non-computer scientists. Thus, modelers do not need to any extra effort to perform model VV&T.

It's inevitable that such a testing framework for ABMS should support these requirements. Towards this objective, we designed and developed RatKit for ABMS to facilitate the model testing process taking into consideration ABMS audience requirements and expectations.

3 Related Work

There has been little work that specifically addresses testing of ABSs and also simulation models.

MASTER is proposed by Wright et al. [19], is a simulation model testing framework for ABSs and compatible with the MASON. MASTER is an external testing tool that provides defining acceptance tests for simulation models. MASTER aims to detect suspicious simulation runs corresponding to the user defined assertions. The modeler defines normal situations, facts, constraints and abnormal situations for the model under test; the framework monitors the simulation runs and evaluates deviations from the normal situations. MASTER is a semi-automatic testing tool and only focuses on prepared simulation models. Rather than developing credible simulation models, it focuses on final VV&T process.

VOMAS, proposed by Niazi et al. [15], is one tool for VV&T of ABMS. They propose using a group of specialized agents; agents specialized in monitoring and testing, over an overlay network to conduct the VV&T process. The agents of the overlay use defined constraints in order to detect unusual behaviors, and report violations if they occur. However, it is not clear how the constraints for the overlay agents are derived and how observations are evaluated. And also, monitoring of the model agents is not clarified. Intervention into the simulation agents breaks the normal simulation run and VV&T gets further away from its main objective.

4 RatKit: A Repeatable Automated Testing Toolkit for ABMS

RatKit (Repeatable Automated Testing toolKIT) is a testing toolkit to facilitate model testing. Testing requires the execution of the model under test as stated in [18]. In this context, each specific model designed for testing is called Test Scenario. Each Test Scenario is defined for specific purpose(s) and includes the required test cases, activities, sequences, and observations. Observations are collected by the Test Environment during the execution of the test scenario. The Test Agent is responsible for evaluating

these assertions according to the collected observations in order to check if these testable elements [18] behave as expected or not.

4.1 RatKit Architecture

The UML model of the RatKit is given in Fig. 1. RatKit uses Junit¹ testing infrastructure for all testing purposes like assertions, test runners, etc. RatKitRunner is the main class for the architecture and the Junit test runner for simulation tests. When a test class is annotated by the annotation @RunWith (RatKitRunner.class) all test methods of the test class are evaluated by the TestAgent. RatKit toolkit is implemented for the Repast simulation environment [23] (RatKit4Repast²).

RatKitRunner first initializes the given test scenario for each test method and creates test scenario elements using RatKitScenarioLoader. RatKitScenarioLoader creates the necessary test scenario files corresponding to the defined test method parameters. RatKit provides test developers to define parametric, periodic and repeatable test executions with the @RatKitTest annotation. RatKitScenarioLoader evaluates the defined parameters for the test scenario and decides the type of test execution. Each RatKitParameter definition corresponds to a simulation model parameter. RatKitParameter values can be constant, number, value iterations like 0 to 100, or a list of values. RatKitParameterSweeper evaluates these parameter definitions and triggers the RatKitRunner for parametric/periodic test scenario executions.

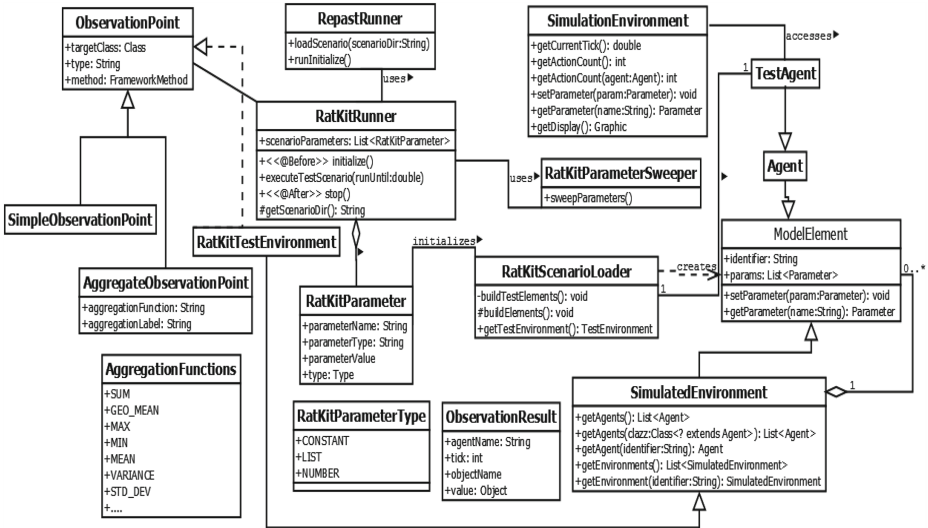


Fig. 1. RatKit architecture (UML class model)

¹ JUnit. <http://www.junit.org> (Accessed: July 2014).

² RatKit4Repast <http://code.google.com/p/ratkit> (Accessed: September 2014).

The test scenario is a sub-model of the model in order to achieve expected behaviors. For simulation tests, each test, corresponding to the testing levels, should have at least one observation point definition. TestAgent executes the assertions corresponding to the observation results. There are two types of observation definitions: SimpleObservationPoint (SP) and AggregateObservationPoint (AOP). SP definitions provide gathering model element properties; a property of an agent or an environment variable. AOP definitions provide summarized results for the model under test using aggregate functions (count, max, min etc.). Each observation point definition is handled by the RatKitRunner and presents it to the TestAgent during the execution of the test cases as an ObservationResult. Each observation result is time stamped, when it's observed and by whom (agent identifier) if required. RatKitTestEnvironment holds the current observation history, a map of the observation results gathered during the execution, and presents to the TestAgent. According to the test execution behavior of the developer, TestAgent executes evaluations (assertions) corresponding to the observations.

5 Case Study: Predator Prey

In this section, we demonstrate the effectiveness of RatKit and its applicability on a well-known case study: Predator Prey. We use a model of wolf-sheep predation [4] of the Repast Symphony [23] that is intentionally simple as an introductory tutorial. While the example is not intended to show real VV&T phenomenon, the model's complexity is high enough to illustrate developing ABMS tests.

This model represents a simple variation of predator prey behavior using three agent types: wolf, sheep, and grass. Both the wolves and sheep move randomly on a grid, and lose energy. The wolves and sheep need to feed in order to replenish their energy, and they will die once their energy level reaches zero. Wolves prey on sheep and may eat them if the two are located in the same spatial position. Sheep may similarly eat grass if the sheep is located on a patch that contains living grass.

In the case study, all of the possible test scenarios are implemented corresponding to our testing levels. It's ready for download in the Ratkit website. Because of page limits we only present a meso-level test: wolf agent prey on a sheep agent. The definition of the test scenario is shown in the Fig. 2. WolfSheepInteractionScenarioBuilder class defines the test scenario. In the scenario, there are two fake agents [18]: FakeSheep, FakeWolf. These agent classes are extended from original agent classes to prevent random movement of the real agent classes. The real purpose of the test scenario is to test the interaction between wolf and sheep agent in the same spatial position. Therefore, both of the scenario agents are located in the same (20, 30) position. We expect at the first tick of the test execution the wolf agent will prey on the sheep agent in the same spatial position.

The test method of the test scenario is shown in Fig. 3. Case study test cases are defined by the wolfEatSheep method which is annotated by the @RatKitTest annotation. The test method annotation includes the definitions of test scenario, execution parameters, simulation model parameters and observation points. In our test scenario, sheepgainfromfood, wolfgainfromfood, wolfreproduce, sheeppureproduce are model

```

public class WolfSheepScenarioBuilder extends Preda-
torPreyScenarioBuilder {
    @Override
    protected void createAgents() {
        Wolf wolf = getEnvironment().fakeWolf("wolf1");
        Sheep sheep = getEnvironment().fakeSheep("sheep1");
        getContext().add(wolf);
        getContext().add(sheep);
        Grid grid = getContext().getProjection("grid");
        grid.moveTo(sheep, 20, 30);
        grid.moveTo(wolf, 20, 30);
    }
}

```

Fig. 2. Test scenario definition

parameters. These are required parameters for the initialization of agent instances and also parameter values which affect agent's behaviors in the simulated environment. Sheepgainfromfood, wolfreproduce, sheepreproduce are constant type parameters. And wolfgainfromfood parameter type is defined as the number (type = NUMBER). In the execution of simulation tests, the parameter value will be increased from 5 to 10 by the RatKit infrastructure.

In this scenario, we intend to test the wolf agent to see whether it gains energy and the sheep agent dies. Firstly, we need to monitor the energy value of the wolf agent. For this reason, we define a simple observation target SimpleAgent class instance (we want to monitor all wolves in the simulated environment) by collecting the values of the “getEnergy” method with the identifier “getLabel” value. Observation results are presented to the developers with an identifier and a time value (in which tick observation result is gathered).

In test cases, we need to separate which result belongs to which agent. In the definition of test scenarios, we define the identifiers of the agents like “wolf1” and “sheep1”.

Another purpose of the test case to test whether the sheep agent has died (removed from the simulated environment). For this reason, we define an aggregate observation point for counting the sheep agent instances in the environment for each tick of the simulation run. The aggregate observation point targets the Sheep agents by using the “count” aggregate function which is named as “sheep_count”.

In that test method body firstly an instance of the RatKitTestEnvironment class is initialized. This class is responsible for presenting observation results to the developers. To test whether the wolf agent gains energy from the eating behavior that is executed in the first tick (tick value is 1.0), we need to get observation results of the “getEnergy” observation results of the initial and final ticks. The wolf agent is created with an initial energy; its energy is decreased by one for each simulation tick and in the first tick the


```

@RatKitTest(runUntil=1, scenarioBuilderClass=
WolfSheepScenarioBuilder.class, parameters =
{@RatKitParameter(parameterName="sheepgainfromfood",
parameterValue="5",parameterType = DOUBLE),
@RatKitParameter(parameterName="wolfgainfromfood",
from="5",to="10",step="1",parameterType =DOUBLE,
type= NUMBER),
@RatKitParameter(parameterName="wolfreproduce",param
eterValue="0",parameterType =DOUBLE),
@RatKitParameter(parameterName="sheepreproduce", pa
rameterValue = "0",parameterType=DOUBLE)},
observationPoints =
{@ObservationPoint(targetClass=SimpleAgent.class,
method="getEnergy",label = "getLabel"),
@ObservationPoint(targetClass = Sheep.class, func-
tion=COUNT,type=AGGREGATE, label = "sheep_count")})
public void wolfEatSheep(){
RatKitTestEnvironment
env=RatKitTestEnvironment.getInstance();
double initialEnergy= env.getSimple( "wolf1",
0.0,"getEnergy");
double energy = env.getSimpleObservation("wolf1",
1.0,"getEnergy");
assertTrue(energy > initialEnergy);
double gain= getParam-
eters().getValue("wolfgainfromfood");
assertEquals(initialEnergy - 1 + gain, energy);
int initialCount=env.getAggregate(0,"sheep_count");
Assert.assertEquals(1, initialCount);
int finalCount= env.getAggregate(1,"sheep_count");
assertEquals(0, finalCount);
}

```

Fig. 3. Test method definition

wolf agent gains energy by eating the sheep agent. These values are evaluated based on the model parameter “wolfgainfood” value.

Another purpose of the test method is comparing the initial and the final sheep count in the environment. For this reason, we get the “sheep_count” observation results of the initial and the final tick value. And we expect here that there are no sheep in the final tick.

6 Future Works and Conclusions

This paper has introduced RatKit and its VV&T approach against ABMS. A tool supporting all needs and aforementioned requirements for VV&T targeting ABMS is an important lack. Our main motivation is filling this gap by the development of RatKit.

Currently using RatKit, users can define simulation tests according to their VV&T purposes. All of the tests are implemented by the users. However, for future work we intend to support automated test case generation from the test scenarios. Most of the testing requirements for the models except domain specific ones have some common points. So, automatic generation of common test cases will be supported by RatKit next versions. In this study, we defined the requirements of ABMS testing frameworks.

Besides, as we mentioned before, a testing framework leading to right design and implementation of ABS models are highly important in order to be able to increase their reliability. For another future work, we intend to define a test driven development methodology for ABMS. Trying to verify, validate and test the ABS models after model building makes ABS development more complex. Such a test driven development methodology that is supported by a testing framework is another gap in the ABMS literature.

References

1. Balci, O.: Validation, verification, and testing techniques throughout the life cycle of a simulation study. In: WSC 1994, pp. 215–220 (1994)
2. Balci, O.: Principles and techniques of simulation validation, verification, and testing. In: WSC 1995, pp. 147–154. IEEE Computer Society (1995)
3. Love, G., Back, G.: Model verification and validation for rapidly developed simulation models: balancing cost and theory. In: Proceedings of the 18th International Conference of the System Dynamics Society (2000)
4. Wilensky, U.: NetLogo Wolf Sheep Predation model. Center for Connected Learning and Computer-Based Modeling (1997)
5. Calvez, B., Hutzler, G.: Automatic tuning of agent-based models using genetic algorithms. In: Sichman, J.S., Antunes, L. (eds.) MABS 2005. LNCS (LNAI), vol. 3891, pp. 41–57. Springer, Heidelberg (2006)
6. Gürçan, O., Türker, K.S., Mano, J., Bernon, C., Dikenelli, O., Glize, P.: Mimicking human neuronal pathways in silico: an emergent model on the effective connectivity. *J. Comput. Neurosci.* **36**, 235–257 (2013)
7. Grimm, V., Revilla, E., Berger, U., Jeltsch, W.M., Railsback, S.: Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *Science* **310**, 987–991 (2005)
8. Epstein, J.M.: Agent-based computational models and generative social science. In: Generative Social Science Studies in Agent-Based Computational Modeling. Princeton University Press, Princeton (2007)
9. Niazi, M.A., Hussain, A.: Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics* **89**, 479–499 (2011). Springer
10. Niazi, M.A., Hussain, A.: A novel agent-based simulation framework for sensing in complex adaptive environments. *IEEE Sens. J.* **11**(2), 404–412 (2011)

11. De Wolf, T., Holvoet, T.: Emergence versus self-organisation: different concepts but promising when combined. In: Brueckner, S.A., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.) ESOA 2005. LNCS (LNAI), vol. 3464, pp. 1–15. Springer, Heidelberg (2005)
12. Sargent, R.G.: Verification and validation of simulation models. In: WSC 2005, pp. 130–143 (2005)
13. Terano, T.: Exploring the vast parameter space of multi-agent based simulation. In: Antunes, L., Takadama, K. (eds.) MABS 2006. LNCS (LNAI), vol. 4442, pp. 1–14. Springer, Heidelberg (2007)
14. Klügl, F.: A validation methodology for agent-based simulations. In: Proceedings of the 2008 ACM Symposium on Applied Computing, SAC 2008, pp. 39–43. ACM (2008)
15. Niazi, M.A., Hussain, A., Kolberg, M.: Verification and validation of agent-based simulation using the VOMAS approach. In: MAS&S at Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW), vol. 494 (2009)
16. Xing, P., Lees, M., Nan, H., Viswanathan, T.V.: Validation of agent-based simulation through human computation: an example of crowd simulation. In: Villatoro, D., Sabater-Mir, J., Sichman, J.S. (eds.) MABS 2011. LNCS, vol. 7124, pp. 90–102. Springer, Heidelberg (2012)
17. Railsback, S.F., Grimm, V.: Agent-Based and Individual-Based Modeling: A Practical Introduction. Princeton University Press, Princeton (2011)
18. Gürçan, O., Dikenelli, O., Bernon, C.: A generic testing framework for agent-based simulation models. *J. Simul.* **7**, 183–201 (2013)
19. Wright, C.J., McMinn, P., Gallardo, J.: Testing Multi-Agent Based Simulations using MASTER (2012)
20. Balci, O.: Golden rules of verification, validation, testing, and certification of modeling and simulation applications. *SCS M&S Magazine* (2010)
21. Beck, K.: Test-Driven Development by Example. Addison Wesley, Vaseem (2003)
22. Gürçan, O., Dikenelli, O., Bernon, C.: Towards a generic testing framework for agent-based simulation models. In: MAS&S 2011, pp. 637–644 (2011)
23. North, M.J., T.R. Howe, N.T. Collier, Vos, R.J.: The Repast Symphony Runtime System, Argonne National Laboratory (2005)
24. Soye, J.-B., Morvan, G., Dupont, D., Merzouki, R.: A methodology to engineer and validate dynamic multi-level multi-agent based simulations. In: Giardini, F., Amblard, F. (eds.) MABS 2012. LNCS, vol. 7838, pp. 130–142. Springer, Heidelberg (2013)
25. Drogoul, A., Amouroux, E., Caillou, P., Gaudou, B., Grignard, A., Marilleau, N., Taillandier, P., Vavasasseur, M., Vo, D.-A., Zucker, J.-D.: GAMA: a spatially explicit, multi-level, agent-based modeling and simulation platform. In: Demazeau, Y., Ishida, T., Corchado, J.M., Bajo, J. (eds.) PAAMS 2013. LNCS, vol. 7879, pp. 271–274. Springer, Heidelberg (2013)
26. Burstein, I.: Practical Software Testing. Springer, New York (2003)
27. Thacker, B.H., Doebling, S. W., Hemez, F.M., Anderson, M.C., Pepin, J.E., Rodriguez, E. A.: Concepts of model verification and validation. Technical report, Los Alamos National Lab., Los Alamos, NM, US (2004)
28. Stasser, G.: Computer simulation as a research tool: the DISCUSS model of group decision making. *J. Exp. Soc. Psychol.* **24**(5), 393–422 (1988). ISSN 0022-1031. [http://dx.doi.org/10.1016/0022-1031\(88\)90028-5](http://dx.doi.org/10.1016/0022-1031(88)90028-5)
29. Carley, K.M.: Validating computational models (1996)

Multi-Agent-Based Simulation XV

International Workshop, MABS 2014, Paris, France, May

5-6, 2014, Revised Selected Papers

Grimaldo, F.; Norling, E. (Eds.)

2015, X, 253 p. 95 illus., Softcover

ISBN: 978-3-319-14626-3