# A Family of Languages for Trustworthy Agent-Based Simulation

Steffen Zschaler
Department of Informatics
King's College London
United Kingdom
szschaler@acm.org

Fiona A. C. Polack
School of Computing and Mathematics
Keele University
United Kingdom
f.a.c.polack@keele.ac.uk

## Abstract

Simulation is a key tool for researching complex system behaviour. Agent-based simulation has been applied across domains, such as biology, health, economics and urban sciences. However, engineering robust, efficient, maintainable, and reliable agent-based simulations is challenging. We present a vision for engineering agent simulations comprising a family of domain-specific modelling languages (DSMLs) that integrates core software engineering, validation and simulation experimentation. We relate the vision to examples of principled simulation, to show how the DSMLs would improve robustness, efficiency, and maintainability of simulations. Focusing on how to demonstrate the fitness for purpose of a simulator, the envisaged approach supports bi-directional transparency and traceability between the original domain understanding to the implementation, interpretation of results and evaluation of hypotheses.

***CCS Concepts:*** • **Software and its engineering** → **Domain specific languages**; • **Computing methodologies** → **Agent / discrete models**; • **Applied computing** → *Computational biology*; *Health informatics*.

***Keywords:*** agent-based simulation, domain-specific modelling languages, CoSMoS

**ACM Reference Format:**
Steffen Zschaler and Fiona A. C. Polack. 2020. A Family of Languages for Trustworthy Agent-Based Simulation. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering (SLE '20), November 16–17, 2020, Virtual, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3426425.3426929


*SLE '20, November 16–17, 2020, Virtual, USA*

ACM ISBN 978-1-4503-8176-5/20/11…$15.00
https://doi.org/10.1145/3426425.3426929

## 1 Introduction

Complexity is inherent to life; most areas of science and policy benefit from an understanding of complex systems. It is a feature of complexity that instrumenting a system in order to experiment on it directly disrupts the natural patterns of interaction [5, 7, 14]: experimental results are at best approximate. There are also significant ethical problems with experimentation: it is ethically undesirable to use animal models, common in medical and biological research, because the experimental set-up disrupts the systems under study and harms (often kills) the experimental subjects. Furthermore, experimentation on live, or recently dead, organisms is not strictly repeatable or reproducible, as the organism and the environment are unique and complex. In a similar way, experimenting on human and engineered complex systems is disruptive and potentially unethical: we cannot experiment safely on economic and social systems (though politicians like to try), or on complex safety-critical systems such as aircraft or chemical/nuclear plant controllers.

Simulation offers a computational alternative to live experiments. If a complex system behaviour can be suitably modelled, then repeatable and reproducible experiments can be run, limited only by computational resources. For a simulation to be trusted, it must be demonstrable that simulation observations are the outcome of appropriately captured behaviours, not experiment artefacts or coding errors.

An example of the trust problem can be seen in recent high-profile agent-based and mathematical simulations supporting research on the COVID-19 pandemic. The simulation designs and code have attracted substantial criticism from the software-engineering community, not least for the lack of recorded rationale (notably parameter value selection). Scientists have declared their confidence that their simulations are sufficient analogues of reality, but this does not amount to demonstrable fitness for purpose, and makes it difficult to challenge and improve the models. Furthermore, the simulations are hard to develop further, reliant on the knowledge and skill of the original developers.

***Principled simulation: the CoSMoS process.*** There are partial solutions to simulation engineering (*cf.* Sect. 5), but the only approach that addresses the whole simulation process from domain exploration and identifying an appropriate simulation focus, to interpretation of results, is the CoSMoS

Process [21]. This is the context for our vision of simulation engineering. CoSMoS presents guidelines and patterns for development and use of fit-for-purpose scientific simulations, and proposes engineering approaches to support interaction between domain experts and simulation engineers.

No techniques or work flows are mandated by CoSMoS, but in projects that have used the CoSMoS approach the modelling has been informal—often using *ad hoc* variants of UML. As a result, whilst developers have used programming environments, none has taken up CoSMoS's suggestion of model-driven engineering. Our vision is of well-founded, tool-supported notations, supporting development techniques that encompass domain and scope exploration, software design and implementation, as well as experimental design and the recording of rationale. The framework exists within CoSMoS; our vision seeks to underpin CoSMoS with a family of domain specific modelling languages (DSMLs).

Crucially, our envisaged family of DSMLs includes languages for expressing expected behaviours and fit-for-purpose arguments, integrated at a fundamental level with executable simulations, as well as languages for specifying appropriate simulation experiments and experimental protocols, with appropriate validation and sensitivity analysis to allow robust conclusions to be drawn. Based on our experience, the vision is expressed for agent-based simulations, but we believe that it would generalise to other forms of simulations.

In the remainder of this paper, we first present a motivating example and highlight some challenges (Sect. 2). Section 3 presents an overview of our vision for a family of DSMLs for agent-based simulation. In Sect. 4 we provide a little more detail of how the vision would apply in reality. Finally, we discuss other efforts for systematic engineering of agent-based simulations in Sect. 5.

## 2 Motivating Example

The motivating example is hypothetical[1] but based on simulator development to support laboratory work at York Computational Immunology Laboratory on (a) formation of Peyer's Patch cell clusters [1, 3]; and (b) granuloma formation in visceral Lieshmaniasis [16]. For each project, an experienced laboratory team worked with CoSMoS team members from engineering disciplines. Four PhDs (and some other student projects), investigated different hypotheses; models and code were developed, exchanged, extended, and abandoned. The series of simulator developments, following the CoSMoS principles, were implemented on Java Mason, Repast and Flame agent platforms [9, 13, 16, 22].

The PPSim (Peyer's Patch Simulator) was developed to investigate laboratory research hypotheses relating to cell cluster development in a neonatal mouse gut. The simulator development involved modelling the domain (cells, chemicals and interactions), validating models with the laboratory scientists, software modelling, and implementation on an agent-based platform, developing bespoke code for component behaviours. The rationale for the development is carefully documented, with argumentation diagrams and extensive text recording the belief in the fitness for purpose of the simulator. Experimentation *in vivo* and *in silico* established new understanding of the triggers to cluster formation (see [1–3]). The collaboration lasted some 6 years. The simulator exists as a downloadable application[2], with associated guidance and scripts for rerunning existing experiments, including full replication of calibration and sensitivity analysis.

The software engineer (SE1) then left. The team now wants to study cluster distribution, and also to use simulation to explore other cell cluster formation. This needs modification of simulation components and behaviours. The team has the ability to modify the domain-modelling for the new hypotheses, but not the expertise to modify the implementation.

A new software engineer (SE2) joins the team. To understand the design, implementation, rationale and fitness-for-purpose arguments, SE2 needs to recreate all SE1's knowledge acquisition. Technical problems arise: missing version histories, historical incompatibility of open-source platforms or code. Inevitably, the simulator documentation does not provide all the detail needed to understand the code: it is clear what the components are, and how they are intended to be implemented, but not how each agent and interaction is coded on the Java Mason platform.

Subsequently, another project wants to adapt the simulation to explore a granuloma formation hypothesis. SE2 starts to map out how to replace PPSim cells with granuloma-forming cells. There are 1:1 mappings between the implicated cell types, but some interaction behaviour and timing parameters (and the cell environment) are not identical. SE2 has solved some of the technical issues, but the code, which comprises the platform representation of agents along with the bespoke coding of interaction triggers and behaviours, bespoke agent environment, and bespoke visualisations and data capture—created by two independent developers with different coding styles—defies systematic modification.

***Key challenges*** arising in this example include:

- The lack of formal mapping between designs and code makes traceability subjective. When design models change, code cannot be simply regenerated.
- The agent platform's agent architecture distributes code for an agent across classes, making it difficult to relate one cell agent to a coherent block of code. This is exacerbated by good programming practices such as creating utility functions for recurrent code and use of domain-specific

[1] In reality, the link between projects and simulators is not so simple.

[2] See kennedy.ox.ac.uk/coles/PPSim

agent hierarchies (i.e., a specific cell is a subclass of mobile cell, which is a subclass of cell).
- The platform code tangles [12] visualisation, data gathering, and computation with agent encodings, meaning that computation cannot be easily modified separately.
- Lack of clarity on how experimental activities are supported in the code. The documentation describes experimental design in detail; experiments can be re-run exactly; but encoding new experiments or parameterisation requires deep understanding of the code.

In short, there is no efficient way to query the domain modelling, the simulation or experiment design, the rationale, or the code base: to find the answer to even a simple question, the enquirer needs to read all the documentation, understand diagrams and code for the simulator and experimentation.

## 3 Overview of Vision

Our vision builds on the CoSMoS Process [21], which proposes three products (referred to, unhelpfully, as *models*):

1. A *domain model* comprises representations of the *relevant aspects of the domain*, expressing the shared understanding of the scientists and software developers. Whilst capturing what is considered relevant, the domain model is incomplete, because some relevant information about a complex system is always unknown or of unknown accuracy. A domain model typically includes representation of hypothesised mechanisms and known observations that need to be recognisable in the eventual simulator.
2. A *platform model* comprises the software design and rationale for the simulator, representing how the domain-model concepts map to computational constructs (*e.g.,* cells to classes). The platform model is used to create and calibrate the simulator. Whilst there is tool support for some techniques to compare and analyse results, to date, all development has used a manual work flow.
3. Whenever a simulation is run, it produces simulation results. These are referred to as a *results model* to avoid the misconception that they tell us something about the simulated reality without interpretation by domain experts.

Because CoSMoS does not mandate a form or level of formality for its products, each project selects its own languages and techniques. Most existing work uses adaptations of software engineering modelling languages, along with some bespoke models, and various forms of text (for assumptions, justifications, rationale and supporting material). To date, all CoSMoS projects have used manual code development. This is a pragmatic approach, but, as the motivating example shows (Sect. 2), manual development inhibits modification, reuse and the reliability of the overall scientific argument. Manually written experiments and scripts are difficult to verify, and there is limited traceabilty from domain concepts, via code, through experimental design and execution to results.

Our vision of a family of DSMLs to support simulation engineering, Fig. 1, would enable incremental transformation, addressing many of the automation challenges.

Our DSML family centres on a fitness-for-purpose DSML that will allow fitness-for-purpose arguments to be captured explicitly and in an analysable form (*cf.* Fig. 1, top two rectangles). We believe that the demonstration of fitness for purpose is the key to trustworthy and scientifically robust computational modelling. Fitness-for-purpose arguments demonstrate that the domain has been modelled appropriately for the stated purpose (justifying modelling decisions based on scientific literature, real-world experiments, *etc.*), that the model adequately reflects reality (by showing that it can reproduce results seen in real-world data), and that the simulation experiments are appropriate to establish conclusions (including, but not limited to, showing that the results establish hypotheses with appropriate statistical rigour). The fitness-for-purpose argument must include explicit hypotheses (*e.g.,* about expected behaviours) and explicit modelling of the simulation experiments to exercise the hypotheses.

Fitness-for-purpose analysis cannot be fully automated. However, an explicit fitness-for-purpose model can maintain links between inputs to, and steps of, a fitness-for-purpose argument, enabling systematic inspection, including by researchers outside the study team. We intend to adapt the existing Goal Structuring Notation (GSN) definition[3] by including explicit, computer-processable links to the other DSML-supported models (and specific versions of these models) that support development and use of the simulation.

The suite of models (diagrams, text, *etc.*) expressing the relevant abstractions of the domain need to be expressed in a domain-modelling DSML so they are accessible to the domain expert. This DSML will likely be adapted to each new research context, to allow natural expression of the domain's specific concepts. However, we anticipate considerable reuse, supported by modular language components—for example, in cellular biology, there are recurrent mechanisms such as gene-regulation networks or energy minimisation principles, whilst cell clustering and cell differentiation have significant generic aspects. A modular approach to language specialisation also makes it easier to extend models and languages.

To further facilitate reuse and extension, we propose to develop the domain-modelling DSML atop a generic agent modelling language providing the basic concepts for agent-based simulation. The generic language will then be specialised for each specific simulation platform (*e.g.,* [15, 17]), enabling automated transformations of generic agent models into platform-specific simulation models.[4]

We envisage that simulation experiments are also designed and modelled via a hierarchy of languages, with a model

[3] www.goalstructuringnotation.info

[4] Many generic simulation platforms already come with a platform-specific simulation-modelling language, often as a library or framework in a general-purpose language such as Java.

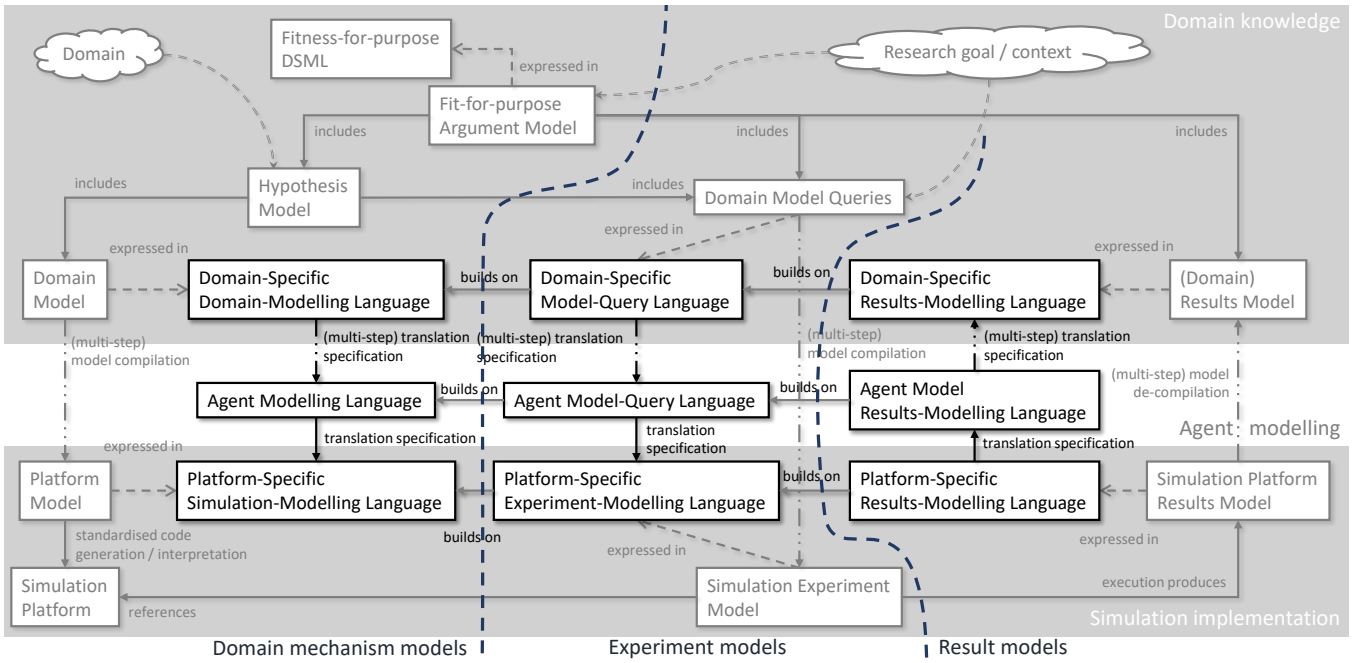


**Figure 1.** Overview of the family of DSMLs. Outer rectangles relate to existing CoSMoS products and techniques. The inner 3x3 structure is the proposed DSML family. Reading top-to-bottom, it moves from domain-focused to simulation-focused, via software engineering modelling. Reading left-to-right, the focus moves from designing, to querying of designs, to results.

querying DSML at the highest abstraction level. Again, such a language has to be specific to each domain, with the potential for reuse across domains, and the use of templates for recurring query types. The query DSMLs should support generation of experiment execution scripts (simulation experiment models) in a stepwise fashion.

Once simulation experiments have been run, the results need to be presented to domain experts for interpretation. This requires translation back from a platform results model (*e.g.,* a log file of a simulation run) to a domain-specific results model expressing results in terms of the domain queries and domain-model concepts, including information about statistical significance. Again, this is a stepwise translation via an intermediary agent-based results model. Results models are referenced from fitness-for-purpose argument models, ensuring the full end-to-end argument is documented. Thus, fitness-for-purpose arguments become live models tracking the current simulation rationale.

Automated generation of executable simulations (*simulation platforms* in CoSMoS and Fig. 1) enables *separation of concerns:* domain experts can focus on expressing their mental model of the domain, whilst software engineers can focus on simulation implementation. Further benefits arising from our vision of a family of DSMLs include the following:

- Automated generation means simulations consistently implement the domain model using well-defined transformations that can be inspected by domain experts and software engineers when maintaining the fitness-for-purpose argument. Because fitness-for-purpose is modelled explicitly, the specific implementation of the transformation can be directly referenced from a fitness-for-purpose argument, enabling complete traceability. This is impossible where simulations are manually developed from domain models.
- In generating simulation experiment models, the automated generator can take into consideration expected boundaries for statistical significance and choose appropriate sensitivity analyses for robustness checking—for example by building on tools such as MC$^2$MABS [10] or Spartan [4]. Again, the generation rules are explicit artefacts that can be referenced from the fitness-for-purpose argument and inspected as needed.
- Building a hierarchy of DSMLs with stepwise translation improves reuse: languages closer to the simulation platform are more likely to be reusable for different types of simulations in different domains. Simulation has been studied for a long time, so simulation platforms are largely stable and do not change substantially. Equally, many domain problems can be simulated using the same fundamental agent-based concepts, but there are problems that will require different concepts to be efficiently simulated. While some aspects of domain models may be reusable across different specific domains (*e.g.,* gene regulation network models), potentially allowing libraries of reusable

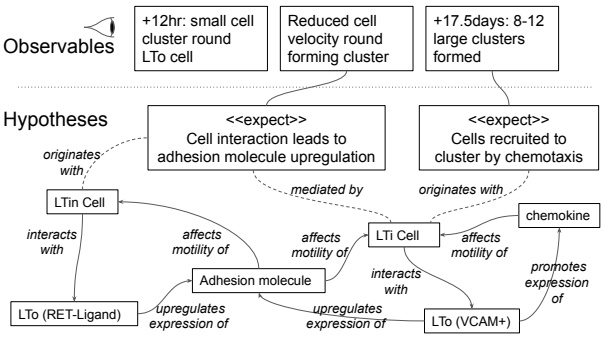


**Figure 2.** PPSim expected behaviours [1], a "hypothesis diagram", devised by York Computational Immunology Labs

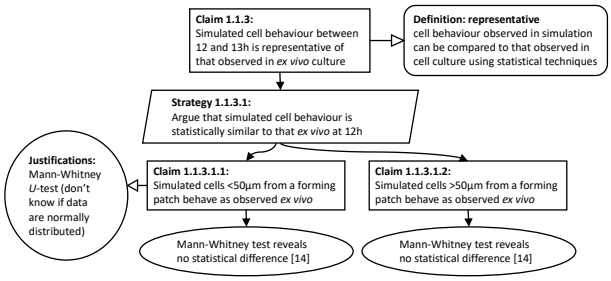


**Figure 3.** Extract of argument that the PPSim domain model is an appropriate representation of the biological domain for this simulation project, using the CoSMoS variant of Goal Structuring Notation [2, 21]

model components to be created, many aspects of domain models may require highly domain-specific languages.
– Stepwise transformation in a hierarchy of DSMLs also simplifies inspecting the transformation specification by domain experts (together with software engineers). Analysing individual transformation steps induces lower cognitive load. Analyses of lower-level transformation steps can be reused; these do not have to be re-inspected every time. Thus, a hierarchical argument for fitness-for-purpose can be constructed, increasing acceptance and trust.

## 4 Application to Example

We illustrate the modelling and validation activities in simulation engineering with diagrammatic models from the PPSim simulator development.

The scope, abstraction level, and purpose of a simulation can be expressed visually. Figure 2 shows a "hypothesis diagram" developed for PPSim. This roughly corresponds to the "hypothesis model" in Fig. 1. The top of Fig. 2 shows three observable phenomena identified by scientists to be recreated in simulation; these would be captured as explicit model queries in our vision. From the observables, two hypothesised behaviours are identified as the simulation focus. Real-world concepts implicated in the observed and hypothesised behaviours are sketched in the lower part of the diagram. This corresponds to the "domain model" in Fig. 1.

In PPSim, the hypothesis diagram was manually translated into state diagrams and then into simulation code. We envision translations to be encoded in automated model transformations.

The rationale for the PPSim domain model is expressed in argument diagrams syntactically based on GSN (*cf.* Fig. 3). However, so far there is no trace link to software engineering language concepts—or, indeed, to expected behaviours models such as that in Fig. 2. Our vision would enable such links to be captured explicitly.

Our vision of a family of DSMLs would allow us not only to link references to concepts across diagramming styles, but also to create queries: for example, we would like to be able to query where concepts such as *LTo* and *Cell* in one of these figures appear in other figures—and in code. Finally, when designing and coding experiments on the simulator, we would like to be able to (a) use well-defined notations and (b) link experimental concepts to those used in the full suite of behaviour and argumentation modelling.

## 5 Related Work

Effective development of agent-based models and simulations has been studied for some time, and model-driven approaches have been explored. As a result, some pieces of our vision have already been studied in various contexts. *However, to the best of our knowledge, a vision addressing the entire argument an agent-based model and simulation supports, from the original hypotheses, to the simulation implementation, the simulation experiments, the calibration and model validation and to the final conclusions, has yet to be achieved.*

So far, model-driven approaches have primarily focused on developing agent languages and (semi-)automated transformations into platform models. For example, the INGENIAS project [6] introduces an agent-modelling language to support replication by enabling the automated translation to different simulation platforms. MAIA [8] is a similar, slightly more recent approach. Here, there is some support for modelling variables of interest and extracting visualisations from the simulation logs. However, this is not connected to hypotheses, simulation experiments, or rationale for developing the simulation in the first place. The OCOPOMO project [19] is, to the best of our knowledge, the first approach that partially addresses the need for achieving traceability from the original domain understanding and research context to the simulation implementation and final data, by allowing the inclusion of hyperlinks to the original data. However, models remain at the agent-language level, thus requiring a mental shift for domain experts to understand the models and relate them to their domain expertise. Other examples of model-driven approaches to agent-based modelling and simulation

can be found in [11] (possibly the first such approach) and [18], a more recent approach that provides a more domain-specific visual syntax for its agent language.

We have emphasised the importance of transparency and explicit modelling for scientific reproducibility and trustworthiness of agent-based simulations. A similar argument applies to scientific process descriptions and our envisioned family of languages could form an integral part of a formalised description of an experimental process. An alternative model-based approach to process description is described in [20].

## 6 Conclusions

We have presented a vision for a family of DSMLs for building robust and trustworthy agent-based simulations, where the models can be understood by domain experts and can be clearly traced to the final simulation and the simulation results, thus constructing an integrated fitness-for-purpose argument. Some parts of the vision have already been explored before. However, the combination of modular DSMLs for modelling domain knowledge, model queries and simulation experiments, and computer-analysable fitness-for-purpose arguments has never been explored. We are currently prototyping such languages in the domains of computational biology and health improvement science.

## References


[1] Kieran Alden. 2016. *Simulation and Statistical Techniques to Explore Lymphoid Tissue Organogenesis.* Ph.D. Dissertation. University of York, UK. http://etheses.whiterose.ac.uk/id/eprint/3220

[2] Kieran Alden, Paul S. Andrews, Fiona A. C. Polack, Henrique Veiga-Fernandes, Mark Coles, and Jon Timmis. 2015. Using argument notation to engineer biological simulations with increased confidence. *Journal of the Royal Society Interface* 12 (2015). https://doi.org/10.1098/rsif.2014.1059

[3] Kieran Alden, Jon Timmis, Paul Andrews, Henrique Veiga-Fernandes, and Mark Coles. 2012. Pairing experimentation and computational modeling to understand the role of tissue inducer cells in the development of lymphoid organs. *Frontiers in Immunology* 3 (2012), 172. https://doi.org/10.3389/fimmu.2012.00172

[4] Kieran Alden, Jon Timmis, Paul S. Andrews, Henrique Veiga-Fernandes, and Mark C. Coles. 2016. Extending and Applying Spartan to Perform Temporal Sensitivity Analyses for Predicting Changes in Influential Biological Pathways in Computational Models. *IEEE Transactions of Computational Biology* 14, 2 (2016), 431–422.

[5] James P. Crutchfield. 1993. *Observing Complexity and the Complexity of Observation.* Technical Report 1993-06-035. Santa Fe Institute. https://sfi-edu.s3.amazonaws.com/sfi-edu/production/uploads/sfi-com/dev/uploads/filer/a7/72/a772b4c5-3c0f-48fb-a56d-2472476dccc8/93-06-035.pdf

[6] Rubín Fuentes-Fernández, José M. Galán, Samer Hassan, Adolfo López-Paredes, and Juan Pavón. 2010. Application of Model Driven Techniques for Agent-Based Simulation. In *Advances in Practical Applications of Agents and Multiagent Systems*, Y. Demazeau, F. Dignum, J. M. Corchado, and J. B. Pérez (Eds.). Advances in Intelligent and Soft Computing, Vol. 70. Springer.

[7] Murray Gell-Mann. 1995. *The quark and the jaguar.* Abacus.

[8] Amineh Ghorbani, Pieter Bots, Virginia Dignum, and Gerard Dijkema. 2013. MAIA: a Framework for Developing Agent-Based Social Simulations. *Journal of Artificial Societies and Social Simulation* 16, 2 (2013). https://doi.org/10.18564/jasss.2166

[9] Richard B. Greaves, Mark Read, Jon Timmis, Paul S. Andrews, James A. Butler, Bjorn Gerckens, and Vipin Kumar. 2013. In silico investigation of novel biological pathways: the role of CD200 in regulation of T cell priming in Experimental Autoimmune Encephalomyelitis. *Biosystems* 112, 2 (2013), 107–121. https://doi.org/10.1016/j.biosystems.2013.03.007

[10] Benjamin Herd, Simon Miles, Peter McBurney, and Michael Luck. 2018. Quantitative analysis of multi-agent systems through statistical verification of simulation traces. *International Journal of Agent-Oriented Software Engineering* 6, 2 (2018), 156–186.

[11] Takashi Iba, Yoshiaki Matsuzawa, and Nozomu Aoyama. 2004. From conceptual models to simulation models: Model Driven Development of Agent-Based simulations. In *Proc. 9th Workshop on Economics and Heterogeneous Interacting Agents.* 1–12.

[12] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-Oriented Programming Gregor. In *European Conference on Object-Oriented Programming (ECOOP'97).*

[13] German Leonov. 2015. *An integrated molecular cell biology and agent-based simulation approach to dissecting microRNA regulatory networks.* Ph.D. Dissertation. University of York, UK. http://etheses.whiterose.ac.uk/id/eprint/12032

[14] Seth Lloyd. 2006. *Programming the universe.* Knopf.

[15] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. MASON: A Multi-Agent Simulation Environment. *Simulation: Transactions of the society for Modeling and Simulation International* 82, 7 (2005), 517–527.

[16] John W. J. Moore, Daniel Moyo, Lynette Beattie, Paul S. Andrews, Jon Timmis, and Paul M. Kaye. 2013. Functional complexity of the Leishmania granuloma and the potential of in silico modelling. *Frontiers in Immunology* 4, 35 (2013). https://doi.org/10.3389/fimmu.2013.00035

[17] Michael J. North, Nicholson T. Collier, Jonathan Ozik, Eric R. Tatara, Charles M. Macal, Mark Bragen, and Pam Sydelko. 2013. Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling* 1 (March 2013). https://doi.org/10.1186/2194-3206-1-3

[18] Fernando Santos, Ingrid Nunes, and Ana L. C. Bazzan. 2018. Model-driven agent-based simulation development: A modeling language and empirical evaluation in the adaptive traffic signal control domain. *Simulation Modelling Practice and Theory* 83 (2018), 162–187. https://doi.org/10.1016/j.simpat.2017.11.006

[19] Sabrina Scherer, Maria Wimmer, Ulf Lotzmann, Scott Moss, and Daniele Pinotti. 2015. Evidence Based and Conceptual Model Driven Approach for Agent-Based Policy Modelling. *Journal of Artificial Societies and Social Simulation* 18, 3 (2015).

[20] Avi Shaked and Yoram Reich. 2020. Improving Process Descriptions in Research by Model-Based Analysis. *IEEE Systems Journal* (2020).

[21] Susan Stepney and Fiona A. C. Polack. 2018. *Engineering Simulations as Scientific Instruments: A Pattern Language.* Springer.

[22] Richard A. Williams, Richard Greaves, Mark Read, Jon Timmis, Paul S. Andrews, and Vipin Kumar. 2013. In silico investigation into dendritic cell regulation of CD8Treg mediated killing of Th1 cells in murine experimental autoimmune encephalomyelitis. *BMC Bioinformatics* 14 (2013), S6–S9. https://doi.org/10.1186%2F1471-2105-14-S6-S9