CHAPTER 1

MODELLING & UNDERSTANDING THE HUMAN BODY WITH SWARMSCRIPT

Sebastian von Mammen[1], Stefan Schellmoser[1], Christian Jacob[2], Jörg Hähner[1]

[1]University of Augsburg, Germany
[2]University of Calgary, Canada

1.1 Introduction

It is well known that generic medical advice and therapy may have adverse effects on the patient's health [9, 53, 55] as well as on the health-care system as a whole, see e.g. [51]. The other way round, individualised treatment has been shown to be more effective, less invasive and reducing therapeutic side-effects [10, 11, 48]. As a consequence, the patient would gain double from an individualised approach to medical treatment. From the technological perspective, individualised medicine can be supported in different ways, for instance by making accurate predictions about the course of a disease or a treatment based on high-fidelity simulation of high-resolution models [8]. It is no less important to convey a comprehensive picture of the state and the development of a patient's health, using visual analytics methods [25] and means of interactive exploration of the physiological processes across the whole human body [21]. The path towards comprehensive computational support for individualised medicine still bears numerous challenges, for instance concerning the legal frameworks, technological limitations on-site, or a lack in computational predictive capabilities. Gradually, various pieces are falling into place that make it possible to retrieve an extensive digital fingerprint of a patient's predisposition and his current condition. Independently, the combination of high-resolution imaging techniques, e.g. based on charged-coupled devices, computerised tomography [18], or magnetic resonance

The Digital Patient.

By Banks et al. Copyright © 2015 John Wiley & Sons, Inc.

author's copy

[3], with the predictive power of large-scale, multi-scale simulation is paving the road for comprehensive individualised medical prevention and therapy. Big strides have been made towards this ambitious goal in terms of important stand-alone benchmarks—for instance in terms of big data analysis [16], predictions at the level of protein interactions [33], or proteome analysis [36]. Yet, in order to comprehensively harness the potential of a *digital patient*, a tremendous need for the integration of diverse technologies remains.

In this chapter, we especially consider the integration of the computational representations used as well as the computational processes taking place during the phases of system modelling & simulation and exploration & analysis. Our according efforts have culminated in *SwarmScript*, an approach to interactively model and simulate physiological systems. We have designed SwarmScript to allow domain experts from the health sciences to translate seamlessly between biological and computational models. The uniqueness of each component of a model has to be properly represented, components be organised into subsystems and systems, and their interactions concerted across all scales. SwarmScript addresses this challenge of large heterogeneous model domains by means of an interaction-based representation and simulation algorithm. It provides a networked, hierarchical view of interdependencies and interactions for model and process analysis. It also enables the amalgamation of extensive model bases into an optimised approximative model during runtime. As a consequence of the diverse requirements that SwarmScript has been built on, it can be presented and understood at different levels: (1) at the formal, representational level, (2) at the algorithmic level, i.e. the execution model, (3) at the level of user interaction, i.e. the description and analysis of physiological models, and (4) at the level of model abstraction. The latter is typically in the hands of the human modeller but increasingly taken over by automated optimisation mechanisms [59, 49, 46].

For the remainder of this chapter, we focus on the user perspective of SwarmScript which provides the best conceptual point of entry to our approach. SwarmScript allows a user (a) to model a particular physiological system and (b) to explore its evolution over time. The process of modelling & simulation (M&S) is iterative—once the workings of a particular model, including its model entities, their parameters and relationships have been understood, the user might want to either refine his model to better match his interests or to alter it to find out more about its complexities [39]. In case this tandem of observation and alteration happens seamlessly and at a fast pace (at realtime), one speaks of *interactive simulation*. Visualisation has been playing an enormous role in making interactive simulations accessible ever since their conception the 1980s [35]. Accordingly, we describe our approach in the light of various renderings immediately taken from SwarmScript runs. In the following section (Section 1.2), we present scientific works related to SwarmScript from the field of agent-based modelling & simulation, focussing on visualisation and visual programming related to SwarmScript. Afterward, in Section 1.3, we introduce the basic vocabulary and how to phrase sentences in the SwarmScript modelling language. This knowledge is put to the task in Section 1.4 where we demonstrate the application of SwarmScript by tracing a previously published agent-based model of the secondary human immune response [45]. In Section 1.5, we discuss the current challenges of SwarmScript and we outline its short-term and long-term potential for accessible M&S of physiological systems. We conclude this chapter with a short summary of our contribution.

1.2 Related Work

Complex systems can be formalised by means of agent-based modelling techniques, whereas the systems' individual parts are represented as (software) agents that interact based on their states and an internal behavioural logic. As this modelling approach is easily comprehensible, also for domain experts outside of the realm of mathematics or computer science, it has received a lot of attention from fields as diverse as economics, the social sciences, and the life sciences [52, 14, 4]. In this section, we introduce preceding works that nourished the conception of SwarmScript. It comprises a basic definition of an agent, and introduces aspects of visual agent-oriented or agent-based programming environments. In particular, it underlines aspects of the formulation of behavioural rules, it outlines the basic vocabularies used by related visual agent-based environments next to their execution models, and it briefly touches on different ways of agent organisations.

1.2.1 Agents and their Representation

Agents representing the parts of a complex system have certain properties and behaviours [62]. The properties typically refer to the data that is stored with an agent, whereas the behaviours describe the system changes that the agent will introduce in specific situations. Accordingly, an agent Ag can be defined as a a quadruple $Ag = \{Sit, Act, Dat, f_{Ag}\}$, whereas Sit is the set of a possible situations, Act the set of possible actions, Dat represents the set of the agent's internal data, and f_{Ag} is the agent's decision function that maps the agent's states to its actions [7]. While this definition allows for arbitrarily complex blueprints of agents, our elaborations focus on reactive biological agents [12] whose behaviours are relatively simple as they do not communicate with each other directly but they coordinate indirectly by changing and reacting to the environment. The value of agent-based models across disciplines [2] has motivated efforts towards accessible agent-based M&S frameworks [40]. To some great extent, their designers have been quite aware of the need for understandable visualisation techniques [29] and accessible user interfaces [26]. As a result, some concurrent agent-based modelling frameworks offer visual programming interfaces that resemble block diagram environments that are also found in modelling toolkits such as LabVIEW and Simulink [24, 6]. In the 1990s, scientists from MIT developed a programmable LEGO brick that allowed to build robots from LEGO parts [43]. The programmable brick connects sensors with effectors, e.g. bumper sensors or light sensors with electronic engines. Researchers were quick to develop LEGOsheets, a user-friendly visual programming environment in which graphical icons representing sensors and effectors were connected to the programmable brick [15]. The behaviour of the programmable brick that processed the incoming sensory data and directed the engines' activity was configured by means of *if-then* rules in a separate editor. LEGOsheets is a specialised visual programming environment based on the more generic AgentSheets framework [41]. Here too, objects are considered agents whose behaviours are expressed through sets of behavioural rules composed of basic operators (conditions such as see or $stacked$ as well as actions such as $transport$ or set). However, neither the application domain nor the application type, e.g. simulations or games, are predefined in AgentSheets. An application model in AgentSheets is composed of several agents, whereas not only the active parts of a modelled system (e.g. E.Coli bacteria) are represented as agents but also reactants such sugar, and even user-interaction elements such as buttons that activate gravity—an according simulation that predicts the waste production of E.coli bacteria in zero gravity is part of AgentSheets' examples library. For configuring operators, AgentSheets offers drop-down

menus to choose from available parameters and agent types. Hereby, it makes extensive use of icons that depict spatial relationships and graphical states of the simulation: For instance, an offset dot in a rectangle depicts an agent's relative position and arrows in eight directions from that dot refer to its adjacent neighbours. SeSAm is another agent-based modelling and simulation environment that offers visual programming [28]. Here, the agents' behaviours are declared in activity graphs, diagrams derived from UML that show the agents' states and indicate their state transitions. In each state, a list of interactions is performed. Since an entry in this list can be another activity graph, SeSAm allows the modeller to define behavioural hierarchies from bottom-up. Some visual programming environments do not model the control flow implicitly through the connections in a diagrammatic fashion. Instead, they provide comprehensive sets of basic programming statements, including those responsible for control flow, as visual jigsaw pieces, whereas fitting pieces imply a syntactically correct sequence of statements. Examples are LEGO/Logo [44], StarLogoTNG [27], and Scratch [42].

1.2.2 Multi-Agent Organisation

Actor-lab presents an approach to agent-based programming of robotic systems that is slightly different from LEGOsheets and its various related software frameworks [61]. In Actor-lab, several agents share and process the information available to a robot and determine its actions. Here, input, agents, control events, and output are visually organised in separate views of the model editor but they are connected with each other to define the flow of information. Typically, sets of agents receive and process sensory signals or controller events and drive the activation of effectors. Again, the agents behave according to sets of rules that are exposed when inspecting the respective model component. The organisation of agents in the Actor-lab environment emphasises the idea of a team of agents that collaborates to reach a given goal, i.e. steering a robot. Alternative organisations in multi-agent systems are, for instance, coalitions which may form, if several agents agree to collaborate on common, temporary (sub-)goals. Groups of agents may also be organised in hierarchical structures that reflect an order of command among the agents and emphasise the higher-level agents' responsibilities as supervisors of lower-level groups. A comprehensive survey on common organisational structures in multi-agent systems is provided in [17]. Hierarchical agent organisations are often used to express their spatial arrangement, which is of great importance given the fact that interactions happen locally [50]. Hierarchical organisations are also a means to integrate multiple spatiotemporal model scales [38].

1.2.3 Designing Interactive Agents

The agent-based modelling paradigm has received a lot of attention also from the field of computer graphics and the entertainment industry. Maya and Blender, for instance, are 3D modelling environments in which 3D meshes can be crafted and equipped with physical properties as well as individual behaviours to produce physically realistic and behaviourally motivated animations [47, 34]. Blender, for instance, offers a visual Logic Editor that allows to model agents and their behaviours. Although such applications provide means to introduce behaviours, they focus on rendering, and the behavioural mechanisms mostly facilitate the production of animations or generative, parametric 3D structures. Modelling interactive agents that can change their environment and be subject to change move into the focus of attention of frameworks targeting the design and development of

games and interactive simulations [32]. Visual programming interfaces are sometimes part of the respective IDEs or can be added as plugins, as for instance the Antares VIZIO Visual Logic Editor for the game engine Unity3D [1]. Primarily, these frameworks provide high-level access to rudimentary physics calculations and computer graphics functions including 3D asset management and scene graph organisation. As SwarmScript draws from this functionality as well, we have chosen jMonkeyEngine for its implementation – an actively-maintained, free and open-source Java-based framework that supports interactive simulation [54].

1.3 Speaking SwarmScript

The development of a domain-specific language promises the basic foundation to integrate various system models into one comprehensive multi-scale simulation of human physiology, as outlined in [21]. The standardisation accompanying the modelling process further allows to deploy automated, self-adaptive model optimisation routines [59]. Its most immediate benefit, however, can be making computational modelling accessible to domain experts with limited knowledge of mathematics or algorithmics. This goal by itself is rather difficult and it requires elaborate decisions about which programming elements, i.e. which data structures and which control flow statements, should be made accessible and in which way. In this section, we recap the agile design process of SwarmScript, which also entails the description of its most recent implementation.

1.3.1 Answering Demand: The Design of SwarmScript

In numerous interviews with domain experts from the health sciences who are closely affiliated with our research groups, we inferred that models are described by means of rule-based behaviours associated to individual biological or chemical agents. This reaffirmed the demand for agent-based, rule-based modelling referenced in Section 1.2. The translation of verbally expressed rules into computational statements is not trivial. Considering a rule to consist of a conditional part and an action, we found that contextual referencing, i.e. referencing specific or unspecific objects or sets of objects in either or both parts of a rule poses a difficult task. Similarly, an agent's ownership of a behavioural rule or of sets of rules cannot be associated easily in general.

1.3.1.1 Graph-based Rule Representation Motivated by grammatical substitution rules that guide the interactions in developmental simulation models [57], we were initially pursuing a graph-based approach to the representation of behavioural rules [58]. As can be seen in Figure 1.1, antecedence and consequence of a rule were represented as two graphs with star topologies of depth 1, encircling a node representing the acting agent. Matching the antecedence in the global state graph of the simulation implied its substitution with the given consequence. The advantage of this representation lies in (a) the inclusion of the acting agent into the behavioural rule and (b) the clear separation between querying and altering the state of a simulation. However, this representation needed considerable modifications to become usable in day-to-day modelling tasks.

1.3.1.2 The Source-Action-Target Triple In order to maintain the clear separation of state queries on the one hand and state changes on the other hand, we introduced a new rule representation, similar to the input-actor-output triple used in Actor-Lab [61]. In particular,

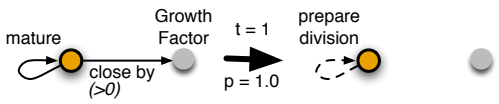
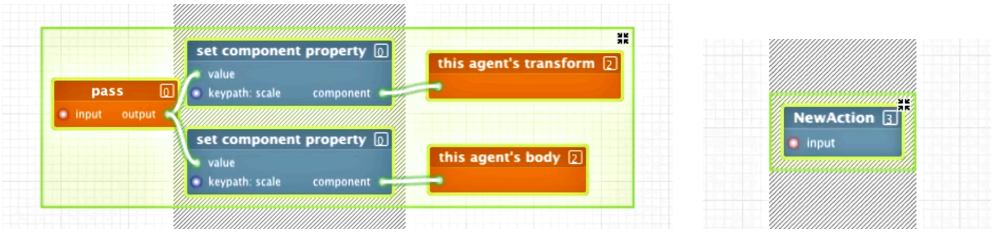


Figure 1.1 One of a set of graph-rewriting rules that describe the proliferation behaviour of a cell.

the next iteration of SwarmScript connected chains of operators (command objects without "needing any detailed knowledge of how the rest of the program works" [5]) that query the given state of a simulation to action operators that would introduce changes to the system. We distinguished between two different semantics of the chains of query operators: Those that provide data to trigger and to inform state-changing actions (*sources*) and those that identify the targets of any of those introduced changes (*targets*). Only if both types of query chains delivered valid results, the associated action operators would be actualised, i.e. their effects be introduced to the simulation. Figure 1.2 (a) shows the trisection of operators: Source queries that process data and trigger actions are shown on the left-hand side of the dashed area. Target queries that identify the objects that would be changed are shown on the right-hand side of the grey area. Actions which cause the change, if both sides deliver proper results are situated on the dashed strip itself. The Source-Action-Target representation of SwarmScript also addressed the need for modularisation [56] (see Figure 1.2(b)), scope (preceding calculations could trigger or skip sections downstream in a chain), and pre-processing data to determine the effected changes. However, although references were explicitly resolved, the ownership of the coded behaviour was not clearly assignable. In addition, the strict separation between source and target queries did not mitigate the inconvenience of potentially redundant expression of references, in cases where sources and targets were identical (which is often the case).

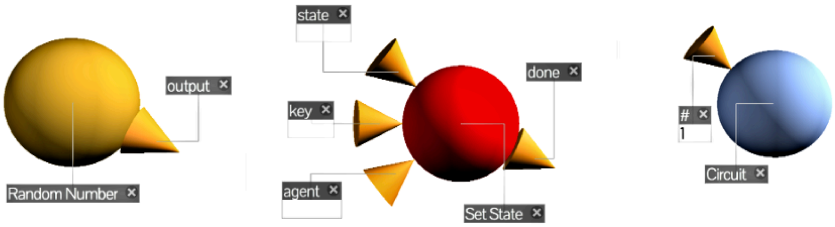


(a) (b)

Figure 1.2 Screenshots from the implementation of the Source-Action-Target representation of SwarmScript. (a) Several operators can be selected (green rubber-band selection) and wrapped into (b) a high-level operator.

1.3.2 SwarmScript INTO3D

The latest version of SwarmScript, SwarmScript INTO3D, lifts the separation between sources and target query chains, it introduces loops denominating an iteration variable for wrappers (now called 'circuits'), and it allows to join actions into sequences to specify an execution order. In order to establish clarity about the ownership of a behaviour, it puts forwards another novelty: The behaviour is projected right into the visualisation of the



(a) (b) (c)

Figure 1.3 Instances of (a) query, (b) action, and (c) circuit operators. The spherical shapes allow for a consistent view in 3D space, the attached cones convey the flow of information between operators. The depictions include label windows that the user can create clicking the respective UI elements. Labels of input connectors contain a text field that presents the currently received input and allows the user to assign a constant value.

simulated world, logical circuitry is embedded in the context of simulated physical and geometrical bodies, relationships are explicitly drawn to potential interaction partners. This projection eliminates the distinction between modelling environment and simulation space. Figure 1.3 depicts instances of queries and action operators as well as of circuit operators. The spheres represent the operators themselves, the attached cones depict input and output connectors, pointing towards and away from the operator spheres' centres, respectively. We decided on spherical operators as they provide consistent visual cues and interaction surfaces, independent of the user's perspective in a 3D modelling/simulation scene. The conic connector shapes intuitively reflect the flow of information between operators—as in functional diagrammatic programming environments, the results of an operator's execution are passed on to downstream connected command objects, where they serve as parameter inputs. Action operators drive the simulation process, as they alone introduce state changes. We visually reflect their important role by depicting them in red. Query operators, on the other hand, which do not affect the simulation state, are rendered in less obtrusive yellow shades. Circuits are visualised in blue as to convey their distinct function of semantically neutral operator containers.

Operators are nested and connected by means of simple drag and drop interactions. A complete behavioural rule is phrased as soon as values/connections are provided for all inbound connectors of an action operator. Figure 1.4 shows a simple example of a random number query being passed into a log action. In general, connectors may maintain $n : m$ connections, whereas the data from its n inbound connections would get aggregated into a collection and its m outbound connections would spread its data.

1.4 A SwarmScript Dialogue

In this section, we present and explore a simple SwarmScript model. Instead of designing an agent-based biological model from scratch, we rely on preceding work by one of the authors which traces the secondary human immune response to infection with the Influenza A virus [19, 20, 45]. Re-constructing an established agent-based biological model, we can direct the focus of our presentation toward the behavioural logic and model visualisation of SwarmScript INTO3D.

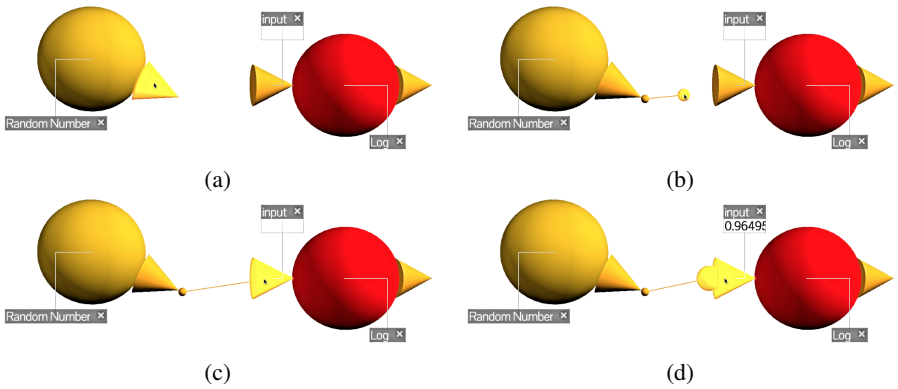
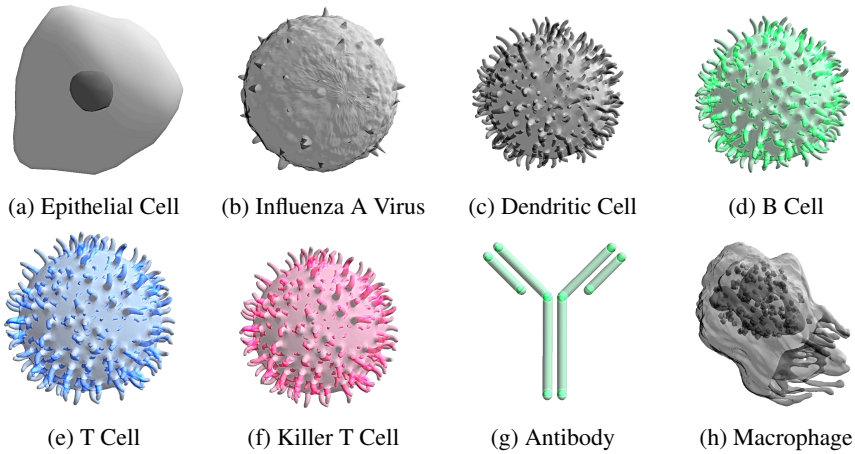


Figure 1.4 (a) Dragging across an output connector creates a new edge, (b) whose head is navigated by the user, (c) to be dropped onto an input connector of another operator. (d) The new connection has established a properly phrased behavioural rule. Therefore, the action operator now receives and processes input information as seen in the input connector's label window.

The domain model itself stages several types of agents (Figure 1.5) that react based on spatial collisions and internal states. Epithelial cells (Fig. 1.5(a)) that constitute lung tissue are susceptible to infection by Influenza A viruses (Fig. 1.5(b)). When infected (at 0.1% chance upon collision with a virus), the epithelial cell is destroyed after an incubation time of 200 time steps and it releases 5 new viruses into its environment. In [45], a comprehensive set of behavioural constants is presented that reliably retrace the progression of the immune response. After exposure to the viruses, dendritic cells (Fig. 1.5(c)) that are scarcely spread across the lung tissue, migrate to the lymphatic system to activate B (Fig. 1.5(d)) and T cells (Fig. 1.5(e)). Some of those mature into cytoxic killer T cells (Fig. 1.5(f)) and destroy infected epithelial cells, terminating the viral spread. B cells boost the production of antibodies (Fig. 1.5(g)) that attach to the viruses – such opsonised viruses are then destroyed by macrophages (Fig. 1.5(h)). Long lived B cells hold the key for a fast secondary immune response releasing great numbers of antibodies as soon as the pathogens are re-entering the system.

In [45], the secondary human immune response simulation was staged in the context of the whole human body. Aiming at interactive multi-scale simulation of physiological processes, this context deems all the more important as two scenes were intertwined – the infection of lung tissue and the recruitment of B and T cells by the dendritic cells in the lymph nodes. In SwarmScript INTO3D, this visual multi-scale view translates into an algorithmic multi-scale perspective. Providing the global context, i.e. the human body, one can dive into the simulation grounds inside the virtual patient's lungs, and recurse ever deeper into any nested components and their behaviours (Figure 1.6).

Each visual object that represents a biological agent (Figure 1.5) is augmented with the according SwarmScript behaviour as seen in Figure 1.7. Floating labels provide clarity about the operators' and connectors' semantics. Configuring the involved operators and combining them into behaviours works directly in three-dimensional space using intuitive drag and drop interaction tasks, as shown above in Figure 1.4. Virtual reality provides virtually infinite space for modelling individual biological agents and multi-scale behavioural modules. In the scope of this chapter, we can only present a rather limited view on the visually modelled behaviours (Figure 1.7). Based on a set of primitive query and action op-



(a) Epithelial Cell (b) Influenza A Virus (c) Dendritic Cell (d) B Cell

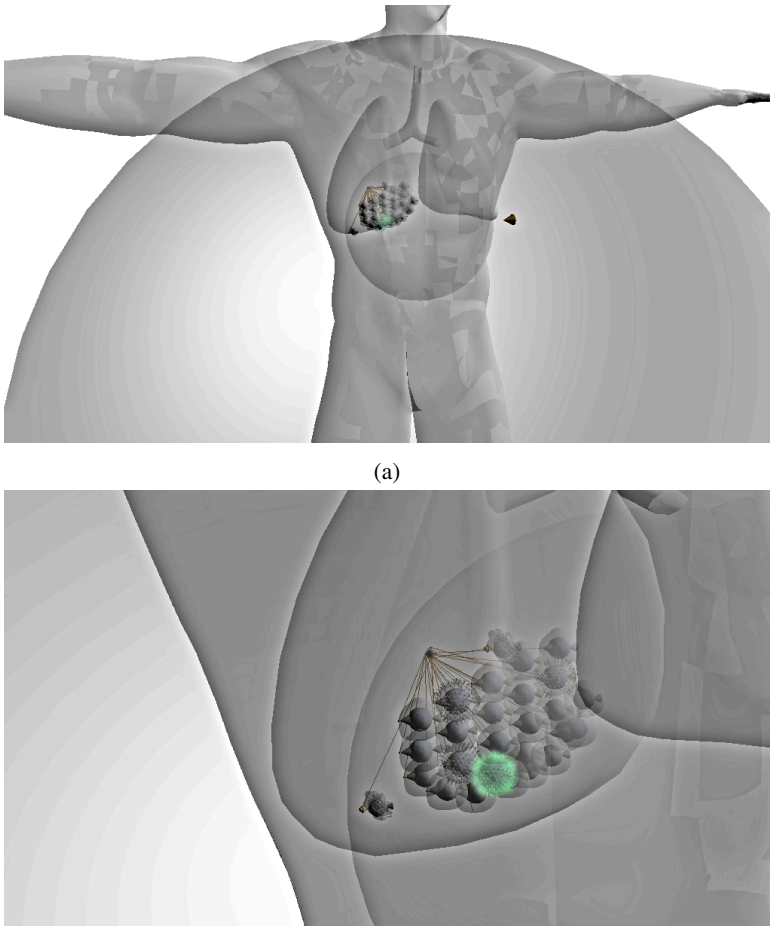
(e) T Cell (f) Killer T Cell (g) Antibody (h) Macrophage

Figure 1.5 Visualisations of the biological agents that drive the SwarmScript INTO3D simulation of the secondary human immune response to Influenza A.

erators, the given domain model has been prototyped: The agents' behaviours are wrapped in circuit operators, they interact through connectors feeding information from and to their environment. Individual operators can be specifically configured and re-used at different locations, as can be modular behaviours, for example the "MoveToClosest" operator in Figure 1.7(f), which calculates the distances to a set of other agents, moves its owner towards the closest neighbour, and yields a boolean flag that indicates whether it has reached its target ("closeToTarget"). The modelled high-level operators can be stored alongside primitive operators for convenient reuse and refinement (also exported as generic XML files). Figure 1.9 shows the according selection window at the centre, next to the heads-up display menu that allows the user to direct the modelling procedures and simulation processes at the top and the view for introspecting an operator on the right. Here, constants could be entered for any inbound connectors, the name of the operator could be changed to match the semantics assigned by the user, and individual connectors could be removed or added to circuit operators.

1.5 Discussion

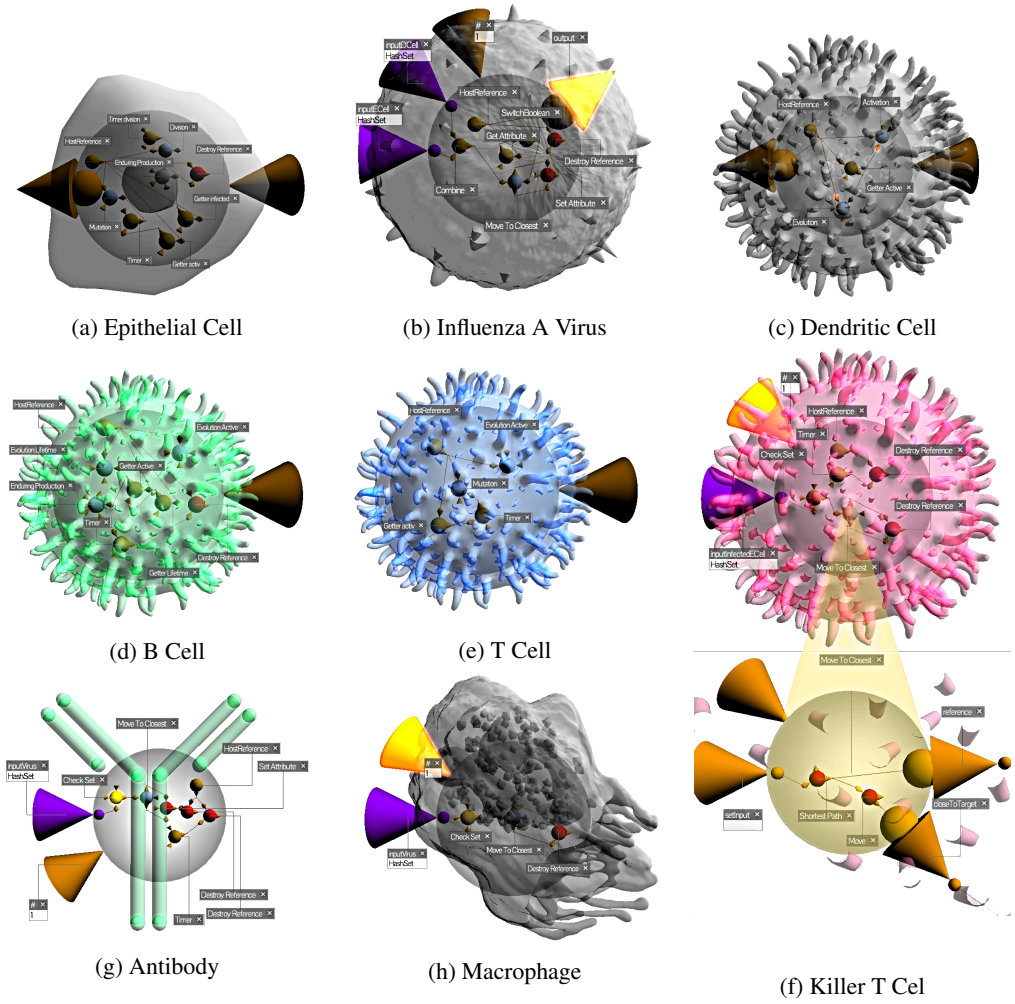
From its graph-based predecessors [58], SwarmScript has evolved into a three-dimensional modelling and simulation approach. It has consequently been extended to meet demands from domain experts, thereby becoming increasingly flexible and expressive. It has also established a close connection between model visualisation and behaviour. However, despite its simplicity in terms of control flow and modularisation, its novelties have also given rise to new challenges. On the one hand, there are challenges in terms of the modelling syntax, its support for semantics, the simulation performance, its support for optimisation. On the other hand, there are challenges associated with visual programming in three dimensions. In this section, we briefly discuss both directions.



(a)

(b)

Figure 1.6 Grey spheres indicate the embedded SwarmScript INTO3D agents and their logic. (a) The lung inside the human body, (b) containing the infected tissue.



(a) Epithelial Cell (b) Influenza A Virus (c) Dendritic Cell

(d) B Cell (e) T Cell (f) Killer T Cel

(g) Antibody (h) Macrophage

Figure 1.7 Combined visualisation and behavioural logic of the biological agents that drive the presented SwarmScript INTO3D simulation of the human immune system. In (f), a nested operator of one of the agents is magnified (yellow overlay).

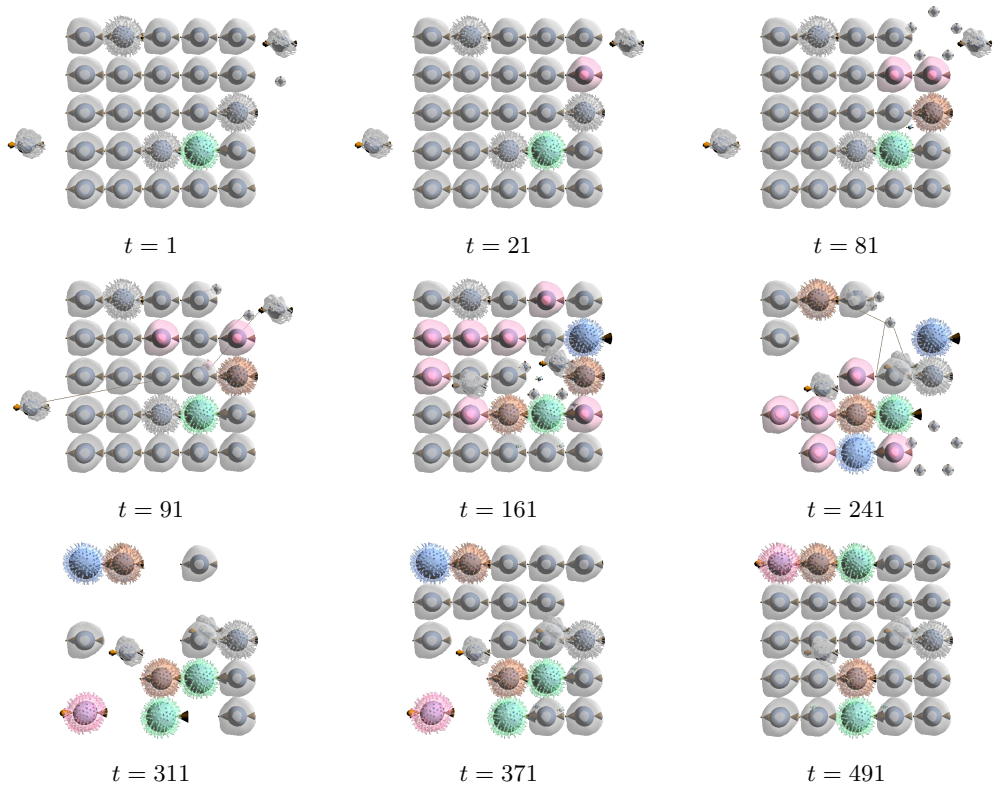


Figure 1.8 The secondary immune response simulation initially hosting twenty-five tissue cells at different time steps t. The progression shows the initial infection, the reaction of the macrophages, the differentiation and recruitment of lymphocytes, the viral spread, the production of antibodies, and the eventual recovery of healthy tissue. Please note that the behavioural logic described earlier is algorithmically and visually translated into temporary relationships and interactions (edges) among the agents. At any point of the simulation, the user can dive into and reconfigure the agents' behaviours.

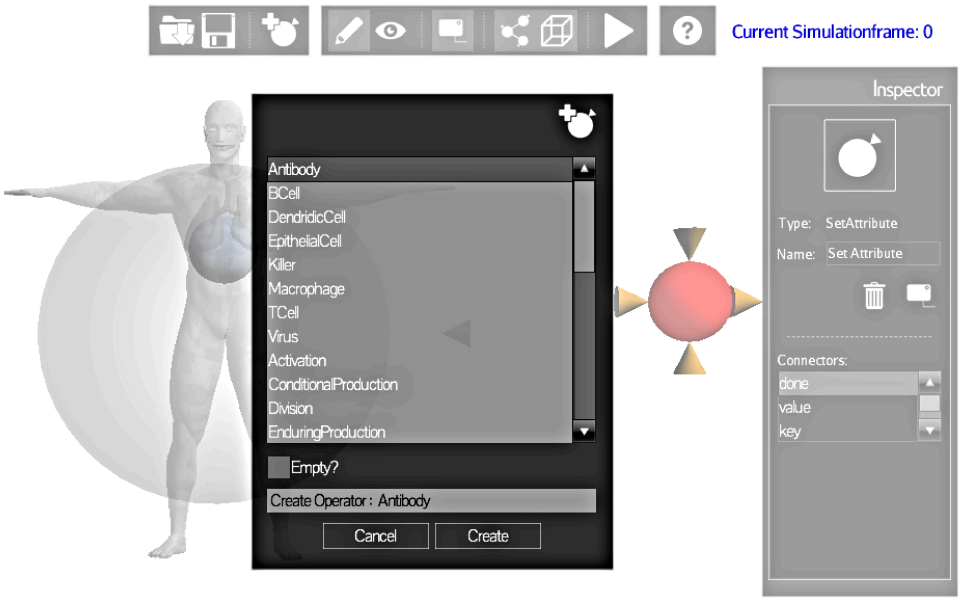


Figure 1.9 The heads-up display for navigating the modelling & simulation phases with SwarmScript INTO3D (top), for selecting and deploying previously stored operators in the current scene (centre), and to configure the currently selected operator (right).

1.5.1 The SwarmScript Language

The goal of SwarmScript is to provide accessible software for developing, presenting and exploring models of interacting biological agents at multiple scales. It is nurtured by the general bottom-up perspective on biological and physiological phenomena, see for instance [50] and [12]. As a consequence, SwarmScript is first and foremost a language that provides the means to express agent-centric, interaction-based behaviour. It differs from similar, agent-based languages in numerous ways, for instance regarding its means to connect operators horizontally and at the same time to allow the construction of modules hierarchically. Only SeSAm addresses these aspects in a similar context relying on UML-based state diagrams combined with lists of activities [28]. In contrast, SwarmScript provides a one-stop solution that amalgamates (a) concrete algorithmic calculations, (b) state-based modelling (via querying and setting state attributes), and (c) the expression of rule-based behaviours. In order to improve the clarity of SwarmScript models & simulations different criteria can be considered to classify its semantics [30]. In SwarmScript, all variables that are queried or modified can be modelled explicitly. For instance, the operator $HostReference$ (e.g. in Figure 1.7(a)) provides an explicit reference to the agent a specific behaviour belongs to. Abiding to a strict guideline for explicit variable usage, the query-action dualism provides a starting point for a clear *axiomatic definition* as only action operators introduce change to the actual system state. In our current implementation, there are still some primitive operators that are reducible to more basic definitions. For instance, the rather basic $SetAttribute$ and $GetAttribute$ operators, which occur frequently in Figure 1.7, in combination with various arithmetic operators could be utilised to

model a $Move$ operator that changes an agent's location based on a given velocity. In this way, a clear *extensible definition* could be established. SwarmScript's strength lies in its *operational semantics* as the execution of interactions can be meticulously traced during the simulation, in terms of conditional relationships, subsequent actions, and state changes. Yet, it could be furthered by the integration of numerous visualisation techniques, including, for instance, a broadly applied colouring schematic for the agents' internal states.

1.5.2 SwarmScript Programming in 3D

Currently, SwarmScript-coded behaviour is projected onto a surface parallel to the camera frustum. This simplifies the user interaction through devices such as monitors and mice. Effectively working in three-dimensional virtual space, e.g. placing operators and connecting them, necessitates the utilisation of novel human-computer interaction methods. This can happen through rendering various depth cues (through shadows and grids on the floor) and projection of the user's body posture into the scene [22], or through immersion of the user into virtual space, for instance, tracking his body and fingers and displaying the scene stereoscopically. The latter full body virtual reality systems could increase the naturalness of the interaction language of the SwarmScript INTO3D interface [13]. They would also allow for the natural exploration of the multi-scale display of the simulated systems—diving in and out of a system, rearranging, rewiring its components. Making the shaping of 3D form accessible (consider early studies [31] combined with novel technologies [60]), they could bridge the gap between modelling of form and modelling of function, and let the bio-agents' physical shape determine their interactions—not only at the microscopic protein-shape level but potentially also in terms of variation at greater levels of organisation, for instance considering variations of organs. Blending between three-dimensional visualisation and behavioural relationships requires the user to navigate in three dimensions for the purpose of modelling alone. This by itself bears several challenges. For instance, the speed of the camera movement relative to the level of magnification has to be adjusted in accordance with the user's needs: Diving several levels deep into a multi-scale model should happen fast, ensuring that the user is aware of the global context. When the user wants to adjust the camera to capture the synchronised interactions of two neighbouring cells, the camera adjustment needs to be very sensitive. When exploring 3D space, it is also important to easily travel and to return to specific locations—all these issues need to be incorporated into a three-dimensional visual modelling and simulation environment as well. Prezi, a vector-based presentation software demonstrates how such "location"-management functionality can be implemented in an accessible fashion [37]. Although three-dimensional rendering conveys a great appeal and naturalness, a hybrid approach to visualisation would be advantageous that makes the model accessible in different modes depending on the information sought [23]. The combination of heads-up displays for navigation and introspection as seen in Figure 1.9 already addresses the need for hybrid display modalities to a very limited extent. Similar to the rich options provided by visual analytics [25], increasing the number of modelling modalities might prove useful in the context of SwarmScript-based models.

1.6 Summary

In this chapter, we provided some background in agent-based modelling and visual agent-based programming, emphasising the often underrated feature of agent interactivity. We

then familiarised the reader with the foundational challenges SwarmScript has been addressing since its inception—starting from graph-based behavioural rules over source-action-target triples to INTO3D. Finally, we demonstrated the mechanics of SwarmScript INTO3D retracing an agent-based model of the secondary human response to Influenza A infection. Based on these elaborations, we discussed the achievements of SwarmScript INTO3D and immediate leeway for its improvement. SwarmScript represents an approach to accessible modelling and simulation of biological agent-based systems. It offers an expressive model representation that originates from a spatial, interaction-based modelling mindset. SwarmScript INTO3D bridges the gap between modelling and simulation spaces, making every model aspect accessible during simulation, providing a truely interactive simulation experience. The design of SwarmScript has been motivated by the needs of a multidisciplinary enterprise. Input from domain experts (teachers, scientists, and practitioners) from the health sciences has informed its evolution, as has research into agent representation, visual programming, and interactive simulation. As a result, it integrates technologies and concepts from a diverse range of disciplines to take the unification of system modelling and simulation one step further towards teachers and students in the health sciences as well as doctors, health-care personnel, and patients.

REFERENCES

1. Antares Universe. Antares universe vizio. `http://www.antares-universe.com/`, March 2014.
2. R. Axelrod, L. Tesfatsion, and K. L. Judd. *Agent-based Modeling as a Bridge Between Disciplines*, volume Volume 2, chapter 33, pages 1565–1584. Elsevier, 2006.
3. P. A. Bandettini. Twenty years of functional mri: the science and the stories. *Neuroimage*, 62(2):575–588, 2012.
4. S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, Princeton, 2003.
5. J. W. Cooper. Using design patterns. *Commun. ACM*, 41(6):65–68, June 1998.
6. J. B. Dabney and T. L. Harman. *Mastering SIMULINK 4*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.
7. J. Denzinger and C. Winder. Combining coaching and learning to create cooperative character behavior. In *CIG*. IEEE, 2005.
8. R. O. Dror, R. M. Dirks, J. Grossman, H. Xu, and D. E. Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annual review of biophysics*, 41:429–452, 2012.
9. I. R. Edwards and J. K. Aronson. Adverse drug reactions: definitions, diagnosis, and management. *The Lancet*, 356(9237):1255–1259, 2000.
10. M. Eichelbaum, M. Ingelman-Sundberg, and W. E. Evans. Pharmacogenomics and individualized drug therapy. *Annu. Rev. Med.*, 57:119–137, 2006.
11. W. E. Evans and M. V. Relling. Moving towards individualized medicine with pharmacogenomics. *Nature*, 429(6990):464–468, 2004.

By Banks et al. Copyright © 2015 John Wiley & Sons, Inc.

12. J. Fisher, D. Harel, and T. A. Henzinger. Biology as reactivity. *Commun. ACM*, 54(10):72–82, Oct. 2011.

13. J. D. Foley, V. L. Wallace, and P. Chan. The human factors of computer graphics interaction techniques. *Computer Graphics and Applications, IEEE*, 4(11):13–48, 1984.

14. N. Gilbert and P. Terna. How to build and use agent-based models in social science. *Mind & Society*, 1(1):57–72, 2000.

15. J. Gindling, A. Ioannidou, J. Loh, O. Lokkebo, and A. Repenning. Legosheets: a rule-based programming, simulation and manipulation environment for the lego programmable brick. In *Visual Languages, Proceedings., 11th IEEE International Symposium on*, pages 172 –179, sep 1995.

16. D. Heller. Combined search in structured and unstructured medical data. In *High-Performance In-Memory Genome Data Analysis*, pages 181–206. Springer, 2014.

17. B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev.*, 19(4):281–316, 2004.

18. J. Hsieh. *Computed tomography: principles, design, artifacts, and recent advances*. SPIE Bellingham, WA, 2009.

19. C. Jacob, J. Litorco, and L. Lee. Immunity through swarms: Agent-based simulations of the human immune system. In *Artificial Immune Systems*, pages 400–412. Springer, 2004.

20. C. Jacob, S. Steil, and K. Bergmann. The swarming body: Simulating the decentralized defenses of immunity. In *Artificial Immune Systems*, pages 52–65. Springer, 2006.

21. C. Jacob, S. von Mammen, T. Davison, A. Sarraf-Shirazi, V. Sarpe, A. Esmaeili, D. Phillips, I. Yazdanbod, S. Novakowski, S. Steil, C. Gingras, H. Jamniczky, B. Hallgrimsson, and B. Wright. *LINDSAY Virtual Human: Multi-scale, Agent-based, and Interactive*, volume 422 of *Advances in Intelligent Modelling and Simulation: Artificial Intelligence-based Models and Techniques in Scalable Computing*, pages 327–349. Springer Verlag, 2012.

22. A. Jaimes and N. Sebe. Multimodal human–computer interaction: A survey. *Computer vision and image understanding*, 108(1):116–134, 2007.

23. M. S. John, M. B. Cowen, H. S. Smallman, and H. M. Oonk. The use of 2d and 3d displays for shape-understanding versus relative-position tasks. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 43(1):79–98, 2001.

24. G. W. Johnson. *LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control*. McGraw-Hill School Education Group, 2nd edition, 1997.

25. D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. *Visual analytics: Scope and challenges*. Springer, 2008.

26. E. Klopfer, H. Scheintaub, W. Huang, and D. Wendel. Starlogo tng: Making agent-based modeling accessible and appealing to novices. In A. Adamatzky and M. Komosinski, editors, *Artificial Life Models in Software*, pages 151–182. Springer, 2009.

27. E. Klopfer, H. Scheintaub, W. Huang, and D. Wendel. Starlogo tng: Making agent-based modeling accessible and appealing to novices. *Artificial Life Models in Software*, pages 151–182, 2009.

28. F. Klügl, R. Herrler, and M. Fehler. Sesam: implementation of agent-based simulation using visual programming. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1439–1440, New York, NY, USA, 2006. ACM.

29. D. Kornhauser, U. Wilensky, and W. Rand. Design guidelines for agent based model visualization. *Journal of Artificial Societies and Social Simulation*, 12(2), 2009.

30. M. Lam, R. Sethi, J. Ullman, and A. Aho. Compilers: Principles, techniques, and tools, 2006.

31. X. Liu, Y. Xiong, and E. A. Lee. The ptolemy ii framework for visual languages. *Relation*, 2(E3):E4, 2001.

32. S. Marks, J. Windsor, and B. Wünsche. Evaluation of game engines for simulated surgical training. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, GRAPHITE '07, pages 273–280, New York, NY, USA, 2007. ACM.

33. Y. Matsuzaki, N. Uchikoga, M. Ohue, T. Shimoda, T. Sato, T. Ishida, and Y. Akiyama. Megadock 3.0: a high-performance protein-protein interaction prediction software using hybrid parallel computing for petascale supercomputing environments. *Source code for biology and medicine*, 8:18, 2013.

34. T. Mullen. *Mastering blender*. Sybex, 2009.

35. S. Narayanan and L. Rothrock. *Human-in-the-loop Simulations: Methods and Practice*. Springer, 2011.

36. N. Neuhauser, N. Nagaraj, P. McHardy, S. Zanivan, R. Scheltema, J. Cox, and M. Mann. High performance computational analysis of large-scale proteome data sets to assess incremental contribution to coverage of the human genome. *Journal of proteome research*, 12(6):2858–2868, 2013.

37. B. E. Perron and A. G. Stearns. A review of a presentation technology: Prezi. *Research on Social Work Practice*, 21(3):376–377, 2011.

38. S. Picault and P. Mathieu. An interaction-oriented model for multi-scale simulation. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

39. F. A. C. Polack. Proposals for validation of simulations in science. In *Proceedings of the 2010 Workshop on Complex Systems Modelling and Simulation*. Luniver Press, 2010.

40. S. F. Railsback, S. L. Lytinen, and S. K. Jackson. Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623, 2006.

41. A. Repenning. Agentsheets: a tool for building domain-oriented visual programming environments. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 142–143, New York, NY, USA, 1993. ACM.

42. M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: programming for all. *Commun. ACM*, 52(11):60–67, 2009.

43. M. Resnick, F. Martin, R. Sargent, and B. Silverman. Programmable bricks: Toys to think with. *IBM Systems Journal*, 35(3.4):443 –452, 1996.

44. M. Resnick, S. Ocko, and S. Papert. Lego, logo, and design. *Children's Environments Quarterly*, 5(4):14–18, 1988.

45. V. Sarpe and C. Jacob. Simulating the decentralized processes of the human immune system in a virtual anatomy model. *BMC Bioinformatics*, 14:1–18, 2013.

46. A. Sarraf Shirazi, T. Davison, S. von Mammen, J. Denzinger, and C. Jacob. Adaptive agent abstractions to speed up spatial agent-based simulations. *Simulation Modelling Practice and Theory*, 40:144–160, 2014.

47. J. Sharpe, C. Lumsden, and N. Woolridge. *In silico: 3D animation and simulation of cell biology with Maya and MEL*. Morgan Kaufmann, 2008.

48. B. Shastry. Pharmacogenetics and the concept of individualized medicine. *The Pharmacogenomics Journal*, 6(1):16–21, 2005.

49. A. S. Shirazi, S. von Mammen, and C. Jacob. Abstraction of agent interaction processes: Towards large-scale multi-agent models. *Simulation*, 89(4):524–538, 2013.

50. A. Spicher, O. Michel, and J.-L. Giavitto. Interaction-based simulations for integrative spatial systems biology. In W. Dubitzky, J. Southgate, and H. Fuß, editors, *Understanding the Dynamics of Biological Systems*, pages 195–231. Springer New York, 2011.

51. C. Steenholdt, J. Brynskov, O. Ø. Thomsen, L. K. Munck, J. Fallingborg, L. A. Christensen, G. Pedersen, J. Kjeldsen, B. A. Jacobsen, A. S. Oxholm, et al. Individualised therapy is more cost-effective than dose intensification in patients with crohn's disease who lose response to anti-tnf treatment: a randomised, controlled trial. *Gut*, 63(6):919–927, 2014.

52. L. Tesfatsion. *Agent-Based Computational Economics: A Constructive Approach to Economic Theory*, volume Volume 2, chapter 16, pages 831–880. Elsevier, 2006.

53. A. Trotti, A. D. Colevas, A. Setser, V. Rusch, D. Jaques, V. Budach, C. Langer, B. Murphy, R. Cumberlin, C. N. Coleman, et al. Ctcae v3. 0: development of a comprehensive grading system for the adverse effects of cancer treatment. In *Seminars in radiation oncology*, volume 13, pages 176–181. Elsevier, 2003.

54. K. Vainer, E. S. Heggen, B. S., N. Hansen, R. Kusterer, R. Bouquet, P. Speed, and B. Owens. The jMonkeyEngine Java Game Engine. `http://jmonkeyengine.org/`, January 2013.

55. J. P. Vandenbroucke and B. M. Psaty. Benefits and risks of drug treatments: how to combine the best evidence on benefits with the best data about adverse effects. *Jama*, 300(20):2417–2419, 2008.

56. J. Vlissides and R. Helm. Pattern hatching: Compounding command. *C++ Report*, April 1999.

57. S. von Mammen and C. Jacob. The evolution of swarm grammars: Growing trees, crafting art and bottom-up design. *IEEE Computational Intelligence Magazine*, 4:10–19, August 2009.

58. S. von Mammen, D. Phillips, T. Davison, and C. Jacob. A graph-based developmental swarm representation and algorithm. In M. e. a. Dorigo, editor, *Swarm Intelligence*, volume 6234 of *Lecture Notes in Computer Science*, pages 1–12, Brussels, Belgium, 2010. Springer Verlag.

59. S. von Mammen and J.-P. Steghöfer. *The Computer after Me: Awareness and Self-Awareness in Autonomic Systems*, chapter Bring it on, Complexity! Present and future of self-organising middle-out abstraction. World Scientific Publishing, in press, 2015.

60. F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.

61. P. Whalley. Representing parallelism in a control language designed for young children. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 173 –176, sept. 2006.

62. M. J. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd, West Sussex, UK, 2nd edition, 2009.