Interactive Self-Organisation

Dr. Sebastian von Mammen

University of Augsburg
Faculty of Applied Computer Science
Department of Computer Science

Interactive Self-Organisation

Dr. Sebastian von Mammen

Submitted in part fulfilment of the requirements for the
Habilitation in Computer Science

Zusammenfassung

Selbstorganisierende Systeme müssen transparent, einfach zu gestalten und zu kontrollieren sein, um einen größtmöglichen Nutzen für Modellierer, Entscheidungsträger und Anwender zu gewährleisten. Um diese Ziele zu erreichen, müssen wir für die Modellierung und Simulation von Systemen mit großer Anzahl von untereinander abhängigen Einheiten neue Perspektiven einnehmen, neue Methoden erfinden, neue Konzepte erforschen und neue Werkzeuge entwickeln. Diese Bestrebungen gipfeln in dem neuen wissenschaftlichen Feld der *interaktiven Selbstorganisation*.

In dieser kumulativen Habilitationsschrift tragen wir zu den Grundlagen dieses neuen Forschungsbereichs bei. Wir motivieren interaktive Selbstorganisation dadurch, dass wir darauf eingehen, wie zugänglich selbstorganisierende Systeme sind, da sie explizit die Brücke zwischen lokalen Interaktionen und globalen Phänomenen schlagen. Indem wir selbstorganisierende Systeme interaktiv gestalten, leisten wir einen wichtigen Beitrag, um komplexe Systeme verstehen und kontrollieren zu können. Selbstorganisation steht dabei nicht nur für eine Eigenschaft jener Modelle, die wir berechnen wollen. Vielmehr verstehen wir sie auch als eine Perspektive, um die Modellierung und die Simulation komplexer Systeme grundlegend zu erweitern. Tatsächlich kann der gesamte Entwicklungszyklus selbstorganisierender Systeme durch Methoden der interaktiven Selbstorganisation bereichert werden. Einige der Ansätze, die in dieser Arbeit dargestellt werden, könnten sich als Vorreiter herausstellen, um computergestützte Simulation in alltäglichen Situationen zugänglich und nutzbar zu machen, ohne dass ihre Anwender umfangreiches Wissen aus der Informatik und der Mathematik an den Tag legen müssen. Diese Art der *individualisierten Simulation* kann nur verwirklicht werden, sofern zugängliche Repräsentationen zur Modellierung in Form aussagekräftiger, interaktiver visueller Darstellungen zur Verfügung gestellt werden. Herausforderungen der Informatik dürfen dabei den Zielsetzungen des Anwenders nicht in die Quere kommen. Vielmehr müssen Methoden der interaktiven Selbstorganisation gewährleisten, dass jeglicher Berechnungsaufwand vor allem dazu dient, jene Fakten zutage zu fördern, an denen der Anwender interessiert ist und jene Probleme zu lösen, die

der Anwender angehen will. Aus diesem Grund muss interaktive Selbstorganisation den Anwendern ermöglichen, ihre Ideen sowohl bottom-up in das System einzuspeisen, also indem der Anwender soviel Detailwissen, wie er als nötig erachtet zur Verfügung stellt, als auch top-down, also indem Beschränkungen und die allgemeinen Ziele des selbstoranisierenden Systems vorgegeben werden. Interaktive Selbstorganisation muss ferner ermöglichen, dass Simulationsexperimente rasch organisiert, ihre Ergebnisse nahtlos untersucht, die involvierten Einheiten und Prozesse gründlich inspiziert und mit großer Präzision verändert werden können. All diese Aspekte der Zugänglichkeit sollten den Anwendern zu jeder Zeit während des Entwicklungszyklus eines Systems zur Verfüngung stehen, sodass die Phasen der Modellierung, der Simulation, der Exploration, der Verbesserung und der Anwendung zusammenfließen.

Die vorliegende Arbeit besteht aus fünf Teilen. Teil I führt das Thema der interaktiven Selbstorganisation ein. In Teil II werden Anwendungsszenarien rund um interaktive Selbstorganisation präsentiert, die breit gefächerte Anwendungsdomänen vertreten - von Kunst über Architektur, Biologie und Medizin bis hin zur Robotik. In Teil III stellen wir Ansätze der interaktiven Selbstorganisation zum Zweck der Modellierung vor. Diese decken vielerlei Aspekte des gesamten Entwicklungszyklus wissenschaftlicher Modellierung und Simulation ab. Nach einem Überblick über den Entwicklungszyklus von interaktiven Simulationen, welcher sich ebenso auf interaktive, selbstorganisierende Systeme übertragen lässt, betrachten wir schrittweise zunächst die Modellierung individueller Verhalten, dann einen Ansatz zur visuellen Programmierung von selbstorganisierenden Systemen im dreidimensionalen Raum und ergänzen diese schließlich mit der Einführung aussagekräftiger visueller Symbole. Die letzten zwei Kapitel des dritten Teils sind eher technischen Aspekten gewidmet, beispielsweise der Verwaltung und Integration grundverschiedener Simulationsalgorithmen (Engines) auf Grundlage sogenannter Komponenten, sowie dem Speichern, Laden und Organisieren von Simulationsmodellen und -experimenten. Jedoch haben wir auch hier, wie bei vielen anderen Aspekten rund um interaktive Selbstorganisation, die technischen Lösungen mit attraktiven und einfach zu handhabenden Schnittstellen versehen. Insgesamt ermöglichen die dargestellten Techniken es, einen Blick in die nahe Zukunft zu werfen, wenn all diese Aspekte unter dem Dach einer einzigen, umfassenden Modellierungs- und Simulationslösung zusammen geführt werden. In Teil IV diskutieren wir Möglichkeiten

der Optimierung von selbstorganisierenden Systemen. Insbesondere betrachten wir Aspekte wie den Übergang von lokalen Verhalten hin zu globalen, systemweiten Eigenschaften. Wir präsentieren Lösungen für Probleme wie das Lenken von selbstorganisierenden Prozessen (Guiding Self-Organisation), wie der Anwender Bedingungen an das System stellen kann, und wie damit einhergehende Erwartungen rechnerisch erfüllt werden können. Die Ansprüche an die Berechnung der Dynamik selbstorganisierender Systeme kann schnell und scharf ansteigen, da es eine Vielzahl von Abhängigkeiten unter den Modelleinheiten geben kann. Deshalb beleuchten die letzten Kapitel des vierten Teils der Arbeit insbesondere die Möglichkeit, den Detailgrad beliebiger Verhaltensmodelle eines Systems zur Laufzeit zu optimieren. Teil V beschließt die Arbeit mit einem Ausblick auf mögliche zukünftige Forschungsziele.

Abstract

Technical self-organising systems need to be transparent, malleable and controllable by human designers, decision makers and users. In order to achieve these goals, we need to gain new perspectives, invent new methods, research new concepts, and develop new tools for modelling and simulation of systems comprised of large numbers of interdependent units. These efforts culminate in the emerging scientific field of *interactive self-organisation*.

In this cumulative habilitation work, we contribute to the foundation of this new field. We motivate interactive self-organisation hinting at the inherent accessibility of the concept of self-organisation as it bridges the gaps between local interactions and global phenomena. By making self-organisation interactive, we take an important step towards making complex systems understandable and controllable. At the same time, we do not only understand self-organisation as a property of the target models that we try to compute, but also as a perspective to advance modelling, simulation and complex system analysis per se. In fact, the complete development cycle of self-organising systems can be shaped by methods of interactive self-organisation. Some of the methods presented in this thesis may be precursors to rendering computational simulation accessible and usable in everyday situations by people without extensive knowledge of compute science and mathematics. This kind of *individualised simulation* can only arise from accessible modelling representations, presented as semantically meaningful, manipulatable visuals. Computational concerns may not interfere with the goals of the users. Rather, interactive self-organisation methods need to ensure that any computational efforts focus on unearthing the facts the users are interested in and on solving the problems the users need to address. Hence, interactive self-organisation needs to empower users to introduce their ideas both from the bottom-up, providing as much detail as they deem necessary, and top-down, specifying constraints and general goals of a self-organising system. It further has to provide the means to swiftly organise simulation experiments, to seamlessly explore their results, to rigorously inspect the involved units and processes, and to concisely manipulate them. All of these aspects of accessibility should be at the users' disposal at any time of the development cycle of a system

to amalgamate the phases of modelling, simulation, exploration, refinement and application.

This work is comprised of five parts. Part I introduces and motivates the notion of interactive self-organisation. In Part II, several application scenarios around interactive self-organisation are presented, covering application domains as broad as art, architecture, biology, medicine and robotics. In Part III, we put forward interactive self-organisation approaches to modelling. These revolve around the whole development life cycle of scientific modelling and simulation. After an overview of the life cycle of interactive simulations, which also applies to interactive self-organising systems, we step from modelling individual behaviours of involved units over an approach to programming self-organising systems visually in three-dimensional space to the introduction of semantically rich symbolic visuals. The last two chapters of Part III are dedicated to rather technical aspects that are usually considered back end technologies such as the management of different computational engines and their combined integration based on the concept of components as well as storage, retrieval and organisation of simulation models and experiments. However, in agreement with all the efforts around interactive self-organisation, we also merge the proposed back end functionalities with attractive and easy-to-use front-end interfaces which allows for a glimpse into the near future when all these aspects will be consolidated by a single, comprehensive and accessible modelling and simulation application framework. In Part IV, we discuss optimisation of self-organising systems. In particular, we tackle aspects such as the bridging from local behaviours and global, system-wide properties. We present solutions to the problems of guiding emergence, of introducing constraints to a system, and how the external expectations can be met. As the computation of self-organising system dynamics can steeply and quickly rise, given that there may be numerous interdependencies between the modelled units, the last chapters of Part IV shed light on the means to optimise the level of detail of arbitrary behavioural system models during runtime. In Part V, we conclude with a visionary outlook on future work.

Acknowledgements

I am appreciative for having been given the opportunity to learn, teach and conduct research at the universities of Erlangen-Nuremberg, Calgary, and Augsburg. I am particularly grateful to the university professors Gabriella Kókai, Christian Jacob, and Jörg Hähner, who have invested in my education and merits, and given me the freedom needed for absorbing the state of the art and for developing my personal research agenda. I would also like to thank my habilitation committee, Jörg Hähner, Elisabeth André and Christian Jacob, for accompanying and supporting the last step of my academic training. Special thanks go to Camille Sinanan and Karen Poloczek who have always stood by my side whenever the burdens of bureaucracy threatened to slow me down.

Since the beginning of my studies more than 15 years ago, I received ample support from numerous fellow researchers that have shared my visions in Computing Science and who have helped realising them, especially by Marcin Pilat, Abbas Sarraf Shirazi, Timothy Davison, Scott Novakowski, Scott Steil, Gerald Hushlak, Josh Taron, Sven Tomforde, and Sarah Edenhofer. Since the beginning of my work at the University of Augsburg, I have also received significant support from many of my students—thank you all so much.

Above all, I want to thank my family for supporting my endeavours. Without the support by my parents, I would have not been able to take an intrinsically motivated academic career path, which I believe bears the greatest transformative potential of all.

Without the love and patience of my wife, Constanze, I would not have learned to pace myself, to take a step back whenever I needed a change in perspective, and to re-energise for the next wave of challenges, for balancing ambitious goals in my career and in our private life.

Thank you all.

To my wonderful wife.

Contents

List of Tables

List of Figures

Part I

Introduction & Overview

Chapter 1

Introduction

Schools of fish, flocks of birds, and social insect colonies—these systems consist of large populations of possibly heterogeneous, mostly simple, reactive agents. The interactions of the individuals may result in system-wide emergent phenomena such as efficient mass transport [11], effective foraging [12], population-wide defence strategies [13] or the construction of complex adaptive nests [14]. The lack of a central control, the decentralised, locally acting individuals, together with the possibility of emergent phenomena render swarms a metaphor for self-organising systems. As such, the swarm metaphor bridges between local interactions and global outcomes, between diversity and homogeneity, between the individual and the population. It highlights the discrepancy and the liaison of different levels of abstraction. Due to the spatial and traceable nature of swarms, this metaphor provides a perspective on scientific models that promises accessibility, flexibility, and scalability of complex systems. Consequently, computational swarms are not only a metaphor for self-organisation but for self-organisation at our fingertips, or **interactive self-organisation**.

Interactive self-organisation describes the effort to making large, self-organising technical systems transparent, malleable and controllable by human designers, decision makers and users. It considers all aspects of the life cycle of according systems and their components. Its development starts with the generation of a domain model, continues with its translation into a platform model, with a subsequent implementation for a specific sim-

ulation platform and it culminates in its evaluation [15, 16]. After stepping through the phases of the development life cycle at least once, the utilisation of the interactive self-organising system may begin in the targeted application context. Its application, in turn, may further nourish and drive the development life cycle by providing ever more elaborate and detailed insights into its numerous functionalities and desirable features.

Each phase of this development life cycle bears special challenges in the context of interactive self-organisation, such as the means to create decentralised, self-organising models that underly certain observed or desirable, global phenomena. The great numbers of interacting units as well as the dynamics that arise from their interactions strongly contrast the development and deployment of monolithic systems. Furthermore, the multitude of interdependencies of the resultant models gives rise to great computational costs which challenges the efficiency and scalability of their simulation. The analysis of the witnessed system dynamics needs to consider both the complexity of the systems' evolutions of state and of topology [17, 18, 19]. Independently of the targeted production system, each of the phases of development, broadly summarised as *discovery, development* and *exploration*, need to be made accessible to the developer/user as he has to devise, implement, test and refine models. In order to minimise the cognitive load of the developer/user [20, 21], interactive visuals should be offered for working with the data at realtime speeds. On top of the development life cycle of products dealing with self-organising systems, it is typically desirable to provide a user interface that empowers to interactively navigate, select, manipulate and control all aspects of the targeted, self-organisation-based production system itself.

Figure 1.1 schematically shows the three phases of the development life cycle alongside the application of a system of interactive self-organisation. For each phase, a self-organising system is depicted, comprised of several interacting entities, or agents, which are exemplarily represented as ants. Arrows between pairs of ants imply a relationship between them. During the discovery phase, by retracing the system's states and its agents' relationships, the modeller/developer strengthens his understanding of the modelling domain and the research context. During the development phase, he devises a model that captures all of the required relationships between the model constituents. At this point, observations about the real-world (grey-dashed arrows

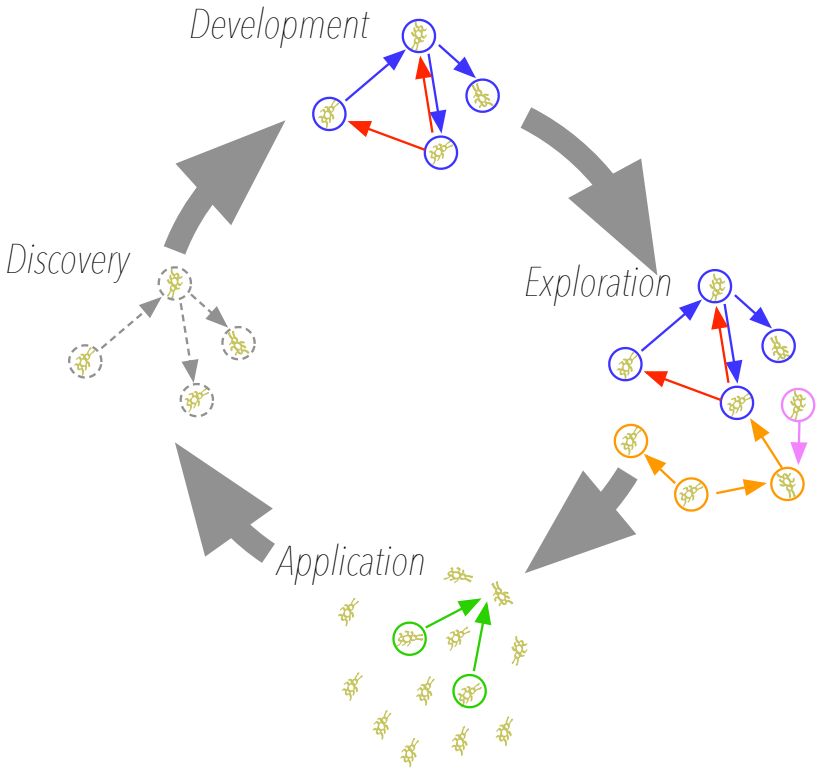


Figure 1.1: The life cycle of interactive self-organisation systems including three development phases *discovery, development* and *exploration* alongside the *application* phase.

of the discovery phase) are translated into a computational representation (indicated by the blue and red arrows). The computational representation inherently preselects, simplifies and discretises real-world processes. In addition, it mirrors relationships that emerge over time and in various situations in a single descriptive instance. This fact is given credit by the differently coloured (blue and red) arrows that represent model relationships that cannot be observed concurrently but become part of one and the same model instance. During the exploration phase, simulations may reveal novel aspects to the modeller/developer that were only implicitly present in the model, indicated by the extended diagram, now also exhibiting orange and pink relationship arrows. Finally, during application, the user of an interactive self-organisation system can harness the previously established models and simulation capacities and directly impact a self-organising system in realiter, for instance by setting the trajectories of individuals of the system as indicated by the green arrows in the last diagram of the development-application life cycle. Areas of application of interactive self-organisation are, for instance, pervasive, networked sensors and computing infrastructures [22, 23], micro- or nanobots for medical procedures [24], swarms of quadcopters for scouting, escorting, surveillance and reporting [25], or interactive, adaptive built environments [26].

1.1 Interactive self-organisation & Simulation

As self-organising systems swarm with large numbers of interacting units, the field of interactive self-organisation aims at researching technologies that support an $n : m$-relationship of n users to m controlled objects or agents, whereas $m >> n$. The approaches of interactive self-organisation, thus, need to make large numbers of objects traceable and controllable by few users or possibly only one user. For the user to understand, visualise and effectively interact with self-organising systems, appropriate models have to be devised first. Second, computational simulations fed with these models can be used to identify and optimise the emergent system dynamics and to maintain real-world self-organising systems within desirable parameter boundaries. The following paragraphs elaborate on this insight, introduce the notion of individualised simulation, the need for agent-based modelling, and the history of interactive simulation.

1.1.1 Individualising Simulation

Simulation can play an important role during each step of the development and deployment cycles of technical systems [27] as it allows one to predict a design's feasibility, the resultant system's functionality [28], and the impact of any changes during runtime [29]. Historically, simulation technologies have interspersed academia and industry, whereas private users have mostly been exposed to them in the context of computer games, for instance in terms of flight and driving simulations [30, 31]. Yet, individualising simulation technology bears the potential of myriad innovations. Centrally issued rules and norms that govern societies can be augmented by individually and independently developed optimisations to adapt to a plethora of local challenges.

Consider, for instance, innovation opportunities in infrastructure and architecture in smart homes [32], the potential of 3D printing technologies [33], or the advantages of individualised medicine [34]. One could harness the power of computational simulations in these and similar transformative fields of application based on understandable, easy-to-use systems to model the status quo, to gain unmatched insight in the given problems, and to receive intelligible advice. Each person could utilise, create and adjust models revolving around their lives and use them to find various optima to drastically improve their life situations—for instance by reducing individual energy footprints [35], by tapping into fresh water supplies [36], or by automated production of artificial limbs [37].

1.1.2 Bottom-Up, Agent-Based Modelling

There is a strongly diversified market for modelling and simulation softwares tailored towards specific application domains. Frameworks such as Mathematica [38], Matlab [39], GNU Octave [40], Sage [41], Magma [42] or Maple [43], define the programmability and accessibility of mathematical modelling and simulation applications. Add-ons provide specific user interfaces and concrete algorithms for rather narrow application scenarios, for instance plugins for EEG analysis [44] or for processing volumetric data [45]. Such add-ons assist the users in focusing

on relatively few parameters important for particular application scenarios, or models. As a drawback, unforeseen alterations or extensions beyond the given parameter spaces are impossible without changing the add-ons' sources, which is typically a rather involved procedure for any code bases, requiring intensive investigations and programming expertise [46].

Here, bottom-up modelling approaches [47, 3, 48], including finite element models [49], particle-based models [50, 51], microscopic models [52], multi-agent models [53], which describe properties and behaviours of individual objects, bear several advantages over modelling at the system level. The most generic way to capture bottom-up models follows the latter, the agent-based model definition, describing an individual as a quadruple consisting of the set of possible situations Sit, the set of possible (internal) data states Dat, the set of possible actions Act and a decision function $f_{Ag} = Sit \times Dat \rightarrow Act$ [54, 55]. Based on this generic definition, an agent can assume different kinds of characteristics and be described, e.g., as passive, reactive, proactive, reflective, etc. Agent-based systems can be altered without great efforts—regarding the numbers of agents, regarding the heterogeneity of agent populations, the state configuration of the overall system, and in terms of individual agents' states and behaviours. Another advantage is the means to directly translate properties and mechanisms of the application domain into the domain model and platform model alike. Due to its flexibility and accessibility, agent-based modelling has received a lot of attention from fields as diverse as economics, social sciences, and life sciences [56].

A considerable number of modelling and simulation systems specifically support agent-based modelling. Examples are NetLogo, RePast, MASON and the Swarm simulation platform: The original NetLogo programme provides an embedded scripting and visualisation environment that lets agents interact on a discrete 2D lattice grid [57]. The RePast simulation suite offers various visualisation and modelling interfaces, including modelling with flowcharts and state charts [58]. It also provides an accessible scripting interface inspired by NetLogo as well as a Java implementation and an optimised C++-based kernel for high-performance simulations. MASON, a discrete-event multi-agent simulation library for Java, provides useful support for 2D and 3D visualisation [59]. The Swarm simulation system focusses on the behaviours and the horizontal and vertical relationships of agents in larger systems [60]. An overview of agent-based

modelling and simulation tools is provided by [61] and [62].

1.1.3 Interactive Simulation

The ease of access to agent-based modelling combined with the great potential of individualised simulations has begun to revolutionise the field of interactive simulations, which is best witnessed by the outstanding successes of game and simulation engines since the early 2000s [63, 64]. They offer modelling 2D and 3D graphics scenes, enrich them with basic Newtonian physics and with scripting of agent-based behaviours in integrated development environments. In the 1960s, interactive simulations entered the stage of scientific endeavours [65]. Early simulation and animation frameworks such as CINEMA [66] rendered simulations in a post-processing step to make emergent effects traceable and transparent. Communicating complex processes especially in the area of operations research motivated the idea to let the user partake at each step of the simulation loop, resulting in human-in-the-loop simulations [67]. SEE-WHY is an according, early visual interactive simulation system [68] which successfully exploited the convincing, often surprising dynamics arising from user interaction with simulation models in realtime.

To summarise the insights of preceding works by Bell [68], Jones [65] and others [67], interactive simulations innovate the field of modelling and simulation regarding the following aspects. In interactive simulations, model specification happens during runtime, while the user is interacting and while the effects are being animated. They provide for a better understanding and evaluation of human behaviours in complex situations such as emergencies. They allow one to consider new aspects, visually highlighting phenomena that are usually not accounted for. Along these lines, they help communicate concepts and challenges, stress inconsistencies and to gain managerial commitment. They generally have a wide appeal and let users participate and shape contents rather than merely watch and consume information.

In the context of interactive self-organisation, interactive simulations become an essential component for empowering people with technology to shape reality—without running simulations

and making predictions about complex system dynamics in realtime, it would not be feasible to understand and interactively control complex self-organising systems in real-world scenarios.

1.1.4 Open, Interactive, Accessible

The swarm paradigm provides a bottom-up modelling perspective that builds on self-organisation and emergence of system properties. Following an agent-oriented perspective, instead of detailed lists of arithmetic operations, the user can focus on descriptions of system components and their interactions at high levels of abstraction. The resulting decentralised networked models are open and flexible, merging modelling and realtime exploration into user-centred modelling & simulation processes, which aptly complements the idea of agile software development [69]. Thereby, the necessary accessibility is warranted not only by a rigorous translation between the actual system and the domain model and according visualisations but also by an explicit involvement of the user, proactively roaming and affecting the modelling space. Numerous examples of such interactive simulation scenarios are provided by 3D multi-agent environments such as breve [70] or Starlogo TNG [71]. Accessibility is thus conterminous with transparency and applicability, emancipating the user from the distinguished art of traditional modelling and simulation approaches.

1.2 Interactive self-organisation for the Wetlab

In an ideal, unbound interactive self-organisation scenario, the user could quickly prototype a comprehensive simulation model, fleshing out spatial details and behaviours of hundreds of thousands of involved agents. Next, he would be given the opportunity to witness the emergence of system behaviours such as cyclic process patterns, branching points, or the convergence of the system state into global attractors [72]. He would also be given the opportunity to automatically repeat and evaluate the simulation within pre-defined parameter ranges, to extract novel insights by learning hypotheses that maximise the information gain [73], and to consider any modelling efforts as only a small, sub-model part of a grander, multi-scale system [74, 75].

The user's efforts would be supported by meaningful, rich visualisation techniques [76, 77] and multi-modal, natural user interaction techniques [78]—in this way, he would be empowered to interact, hone and explore the system model in any desirable ways, posing as little cognitive and motor-sensory challenges as possible [20]. Novel augmented reality technology including head-mounted devices, eye and finger tracking sensors would bridge the gap between simulated predictions and real-world systems, providing invaluable data for learning, decision making and guiding the user's actions [79, 80].

In the context of biological developmental processes [81, 82], for instance, the ideal simulation workbench would allow a modelling entry at the intercellular level, offering the means to model layers of mesenchymal and epithelial tissues, empower individual cells with the capabilities to adhere to each other, to divide, to migrate, to produce and emit morphogens etc. [83]. The model would relate these foundational operations to time, to biochemical or biophysical signals such as the diffusion of homeobox gene concentration [84] or mechanical forces [85]. Based on such intercellular interactions, morphological processes would emerge, shaping anatomy and physiological infrastructure of developing organisms [86]. Fast forwarding in time, the obvious effects of morphology-affecting developmental processes would wane, a metabolic equilibrium would establish itself. The model could be extended to provide more facts at different levels of scale [87], for instance by detailing the production pathways of signalling molecules or by introducing materials that define the cell's structural properties [88]. Similarly, empirically identified emergent properties such as the cell's surface tension, or its adhesion coefficient, could be superimposed, the parameters of the lower modelling levels be automatically adjusted top-down, resulting in a consistent, self-adapting middle-out model [89]. At any point in time, disruptions of the developmental processes could be explored, the formation of anomalies could be traced and countered with minimally invasive treatments, without loosing sight of side-effects at all conceivable scales of the organism's definition.

The tandem of in-vitro and in-silico experiments would ensure the validation of each component of the model and the simulation, respectively, resulting in a profoundly accurate model and providing clear perimeters of the experiments' outcomes and the simulations' predictive powers. A sophisticated, accessible and flexible augmented reality interface could mediate between in-

vitro and in-silico models, allowing developmental biologists to setup and experiment relying on standard assay procedures. The scientist's activities would be supported and guided by the augmentation of in-vitro experiments and the projection of in-silico simulations, imparting all the benefits of computing technologies, including virtually limitless resources, the possibility to go back and forward in time and to venture into new exploratory directions. Depending on the application domain, the wide-spread adoption of swarm-based modelling and simulation could also lead to far-reaching model improvements that could accelerate overcoming the gap between in-vitro and in-vivo predictions.

1.3 Research Challenges

Several steps need to be taken in order to arrive at the outlined application scenario of interactive self-organisation for the wetlab. They are as broad as devising a standardised representation for self-organising system models, identifying and honing visualisation and interface methods, and maintaining computability despite large degrees of freedom.

1.3.1 Representation & Standardisation

Devising and establishing an expressive and broadly deployable representation for self-organising systems is an important step for building interactive, large-scale, open systems. Current generic approaches to modelling self-organising systems typically do not constrain the behavioural definition of the agent implementations but allow for unconstrained algorithmic definitions, again consider, e.g. [70, 71]. This leaves the responsibility to the modeller to clearly define the scope of the agents' knowledge and the degree and the effect of their interactions. Although convenient for the modellers at first, the lack of a formal foundation often implies poor means of scientific scrutiny and accordingly low significance and transferability of any results.

The similarity of representations utilised by researchers who diligently align their implementa-

tion work with formal descriptions, including conditional rules [90], condition-action pairs [91], subject-predicate-object triples [92], or numeric decision functions [93], may have concealed the view on a general formal representation. Especially, since in complex systems—and self-organising systems with large numbers of interdependent units tend to qualify as such [94]—subtle differences in data management and its interpretation may inadvertently result in dramatically different results, as for instance investigated in the context of motion of Brownian particles [95].

Therefore, the broad adoption of a specific behavioural representation to layout the interactions among self-organising agents in combination with a simple execution algorithm—speaking of solving or simulation would be a gross overstatement in this context—would be a great advancement towards scalable interactive self-organisation. If the representation further allowed for extensible definitions, axiomatic primitives could provide a foundation to build standardised high-level operators for various application domains. The accompanying denotational and operational semantic richness would be an important step towards the broad and fruitful dissemination of self-organising system models and their simulations.

1.3.2 Visualisation & Interfaces

A well-defined representation answers the need for an extensible, consistent formal foundation. It enables the application of rigorous analyses and proofs and ensures the possibility of a scientific discourse as provided by traditional, mathematically expressed models and theories. Additional efforts need to be made in order to support modelling, simulation and application of self-organising systems interactively. New methodologies for visualisation and interaction need to address novel aspects including inherently open interaction topologies, the focus on emergent effects and interwoven, flexible interfaces of agents, which also promote the tight integration of user-centred interferences at arbitrary phases of model building and simulation. Therefore, in addition to formal standardisation, an equally generic definition of an application programming interface is needed to provide the foundation for new visualisation methods and human-computer interaction processes. First explorations towards corresponding human-swarm interfaces, which are tailored towards application-oriented domains, such as robotics, have

already been made, see for instance [96, 97, 98, 99].

The discrepancy between the individuals' and the collective's states and behaviours, or, respectively, between any high-level goals and low-level, individual control, remains a challenge that builds on a diverse set of research questions, including the translation between different degrees of abstraction [100] as well as the inverse problem [101, 102], which aims at inferring local rules based on global requirements. The user will benefit most, if beautiful visualisations can be found to effectively communicate these relationships, wrap them in attractive looks and, at the same time, decrease the cognitive load of the user [103].

Especially graph and network based visualisation techniques seem appropriate to communicate states, relationships and interactions among large numbers of simulated units. They offer a means to reflect the relationships in self-organising models and a means to investigate hierarchically organised clusters instead of solely focussing on individuals [104]. Next to concrete methods of visualisation, effective interaction demands for simplicity and consistency regarding the interfaces to models and simulations. Traditionally, there have been separate interfaces for visualisation and editing with respect to modelling/design, simulation, exploration and analysis. Visual programming with flow-based visual languages came up in the late 1960s [105]. Since then, it has permeated through a vast number of fields of application [106, 107, 108]. Generally speaking, visual programming merges visualisation and editing environments. Object-oriented visual programming has been discussed on multiple occasions, e.g. [109] and [110], and so have been agent-based visual modelling environments [111, 90]. The according interfaces build increasingly stronger links between the model-building interfaces and the simulation inspectors, for instance by utilising icons of simulated agents when visually composing their interaction rules [111]. Taking this idea one step further and projecting the visual editor into the actual model space representation, the interfaces for modelling and simulation can be merged, unified and their accessibility be improved. Especially in the context of interactive self-organisation, this convergence is of great importance, as both model and simulation need to be projected into real-world spaces, as for instance in [112].

1.3.3 Scaling D^2S

Given a rigorous formal foundation, a standardised programmatic methodology, accessible means of interaction and effective, beautiful means of visualisation, one major obstacle would still keep interactive self-organisation strongly limited, namely computational complexity.

Due to the great degrees of freedom bestowed on the individuals in self-organising systems, including potentially changing interaction topologies, the arising swarm simulations can be considered dynamic systems with dynamic structures, or D^2S, the most volatile and therefore most challenging class of complex system representations [113, 114]. In the context of optimisation, such tight coupling of state space and the space of interaction topologies may alter the space of possible solutions during the actual search for better results. This phenomenon is also referred to as a self-referential fitness landscape [115].

Often, the actual costs of D^2S simulations are far from the worst case complexity, as the agents' states may remain fixed among certain subsets of agents and over long periods of time. Equally, their interaction topology might, to a great extent, remain static. However, even in models as simple as virtual flocks of birds that concert their collective movement based on the individuals' relations to their neighbours [1], the resultant computational complexity takes its toll. Accordingly, various means of efficient calculations have been investigated, including porting such flock simulations to graphics processing units (GPUs) or applying spatial data structures to reduce the complexity of continuously re-calculating the interaction topologies [116].

Although other swarm-based models may not exhibit the same degree of spatial interdependence between the individuals' states and their interaction topologies [117], they all suffer from the potentially great computational complexity of D^2S. Excluding biological interdependencies, focussing on the dynamics of physical states and spatial relationships, this challenge is similar to the n-body problem for predicting celestial trajectories based on mutual gravitational influences [118]. Next to exhaustive $\mathcal{O}(n^2)$ calculations for rather accurate solutions, the existing body of works dedicated to this problem also suggests approximations based on spatial data structures

that allow for well-informed pruning of less significant interactions [119]. Pruning based on spatial data structures also represents the standard approach to solving for collisions between geometric objects as used in realtime computer graphics and physics environments [120].

These examples show that in order to scale system models revolving around spatial interdependencies, spatial relationships are categorised, partitioned and exploited. Considering that spatial relationships are but one, albeit often important, aspect of interactive self-organisation, the link between operational semantics and optimisation opportunities needs to be generalised. Any state that cannot be captured spatially, including, for instance, the number of cycles a cell has already passed through, the adhesive force towards its neighbours, or its proteomic configuration, may lead to system attractors that can be exploited for efficiency reasons, too.

The great degrees of freedom inherent in the individuals of computational swarms are partially responsible for the generated computational costs. Yet, they are a key ingredient for defining models that can exhibit valuable insights in the emergence of system-wide states. At the same time, the occurrence of emergent phenomena, the convergence of swarm systems into attractor states and cycles, also provides a semantic lever for applying dearly needed model optimisations [121]. In particular, new insights in the evolution of an open system can help to constrain, and thereby simplify, its underlying model's freedom. The automation of this process based on demand, i.e. to simplify whenever computationally necessary and to relax the model complexity whenever semantically relevant, mirrors the scientific method applied by humans for centuries to find out and utilise facts about reality. Only now, it is designed to automatically adapt and to help reveal information beyond current knowledge through large-scale swarm-based computational simulations.

1.4 Scope of this Work

This habilitation work cumulates research efforts that contributed to the field of interactive self-organisation within the past six years. The greatest challenge so far has not been of technical nature, for instance about representation, interfacing or scaling. But it has been a challenge

of context and interdependencies: Hardly any of the technical challenges could have been effectively investigated by itself. Without considering the means of representation, a generic way of automated model abstraction and optimisation could not be innovated. Without considering the users' needs with respect to their modelling abilities and their modelling domains, effective representations, visualisations, interfaces, and optimisation algorithms could not be devised.

Therefore, the cumulated works presented herein revolve around various application domains, such as medical education, biological research, art, architecture, or robotics. Concrete application domains are presented in Part II. The next chapter will conclude Part I, providing the background and the overview of all the presented works. Part III will focus on the aspect of modelling of interactive self-organisation and touch upon topics such as model and simulation development, graph-based representations, modelling in 3D simulation space, visual behavioural programming of agents, and data management for interactive self-organisation. Part IV will focus on aspects of optimisation of self-organising system models—either in terms of model parameter optimisation or in terms of adaptive model optimisation during runtime.

Next to fleshing out and investigating concrete interactive self-organising systems, modelling and simulation techniques of interactive self-organisation represent the lion's share of the included chapters. Naturally, each of them focusses on a specific aspect, such as graph-based representation, visual programming, or abstraction of agent interactions. Yet, these works have been cross-fertilising identification of demand, requirement specifications, and concepts at different levels of abstraction. The next chapter details the interplay of these cumulated works.

Chapter 2

Overview

Considering the multi-facetted goal of interactive self-organisation—namely making large, self-organising technical systems transparent, malleable and controllable by human designers, decision makers and users—it is clear that a considerably large research community contributes to this field through novel algorithms, techniques and application scenarios at various levels of abstraction and tackling various perspectives on the subject matter. In this habilitation thesis, published research works are cumulated that consider three concrete perspectives on interactive self-organisation: Part II of this thesis presents several publications with a focus on application scenarios that exemplarily show where and in which ways interactive self-organisation matters. Part III introduces different aspects relevant to modelling interactive self-organising systems. Part IV reflects on the need for optimisation in the context of computing interactive self-organisation systems. In this chapter, we briefly summarise these respective parts' contents. We elaborate on their relationships and draw the big picture on how these contents require and stimulate one another and how they might eventually be considered small steps towards ubiquitous, empowering interactive self-organising systems.

2.1 Overview of Part II: Application Perspectives

The presented application scenarios reach into the following domains: Medicine, developmental biology, architecture, art, power grids, ecology, and robotics. The goals of the respective interactive self-organising systems are: Research, education, training, planning, construction, maintenance, exploration, and entertainment. Figure 2.1 classifies the chapters of Part II of this thesis with respect to application domains and goals, whereas the goals are abstracted to the notions of design, training, and research represented by the three corners of a triangle. The contrasting, yet complementing goals of training and research can easily be understood to occur on opposite ends of the spectrum. The third overarching goal, namely design, results from the fascination by the author to manipulate and reinvent reality. Research generates and validated knowledge, training disseminates and internalises knowledge, and design can be seen as an outlet channel that allows to transform reality based on knowledge. Hence, while all three displayed goals are addressed by research studies targeting the application of interactive self-organisation, they complement each other, and in parts, they also build on each other. We tried to structure the presentation of the respective research works keeping these relationships in mind.

First, in Chapter 3, we introduce work on the Lindsay Virtual Human, a visionary project aiming at the full realisation of interactive self-organisation for research and training in the life sciences. The subsequently presented works address two aspects in more detail: (1) The simulation of physical interactions in virtual environments (staged in the context of dental operations) in Chapter 4, and (2) the modelling and design process arriving at computational representations to retrace biological developmental processes in a self-organising, agent-based manner (Chapter 5). The respective chapter is followed by an overview of self-organisation models crafted by architecture students (Chapter 6). The rationale bridging developmental biology and architecture is rooted in the fact that both disciplines consider construction processes and resultant artefacts. Looking at myriads of biological, chemical, and physical interdependencies that result in robust, adaptive development of organisms may, by a long shot, exceed

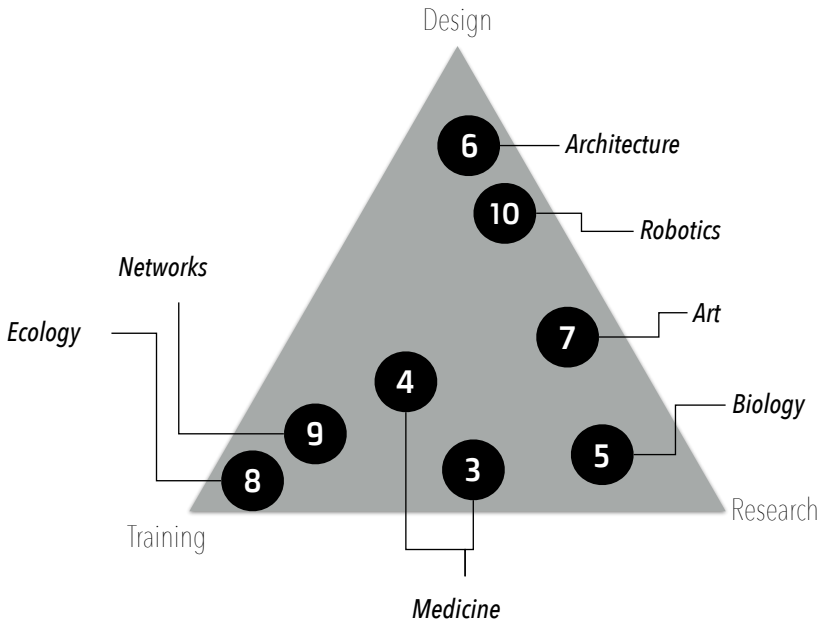


Figure 2.1: The content domains of the Chapters (numbers) of Part II of this thesis with respect to their goals: Training, Design, and Research/Exploration.

the complexities handled by human architects but the very same principles of additive and subtractive construction apply. Moreover, architecture also considers a multitude of objectives next to the generally known criteria of structural integrity and energy efficiency—building usage, its users' experience, legal constraints, integration into the built environment are examples of additional factors that are considered by architects, also relying on interactive models and simulations of self-organising systems. The architectural perspective is followed by an academic excursion into the fine arts in Chapter 7, where three artists explore the dynamics of interactive self-organising systems relying on established art media. The playful artistic exploration of self-organising processes is followed by two examples of so-called serious games, in which the user is given the opportunity to interact with self-organising systems. In the first, presented in Chapter 8, the user is entrained to maintain the ecological balance in an aquarium. In the second (Chapter 9), he builds up experience in setting up and modifying power grid infrastructures. We conclude the series of exemplary application scenarios of interactive self-organisation with Chapter 10, presenting an overview of OCbotics, where, based on the principles of Organic Computing, self-organising robots and robot ensembles are designed to dynamically react and self-adapt to changing goals and situations. These robotic systems interface with their users to receive, depending on the situation, high-level goals or detailed low-level instructions.

In the following paragraphs, we elaborate more in-depth about the contents of the respective chapters/publications and how they contribute to the field of interactive self-organisation.

2.1.1 The Lindsay Virtual Human Project

Chapter 3 presents the early efforts by the Lindsay Virtual Human project to make medical contents accessible through visualisation and interaction. These efforts comprise the design and development of interfaces to navigate and explore the human body at different spatial scales. In addition, the Lindsay Virtual Human project researches model representations, modelling languages and interfaces, runtime environments including rich content management options, including fast storage and retrieval, automated versioning, distribution and optimisation techniques apt for making physiological processes computable in realtime and to making them accessible in interactive ways. These processes describe, for instance, the secondary response of the immune system, blood coagulation in the context of the circulatory system, or biochemical propagation of electric charges in the nervous system. The variability of these processes is exposed to the user, allowing him to interact within broadly defined parameters.

The interactivity the project aimed at, rendered traditional, solely equation-based system representations inappropriate, as the degrees of freedom would have been drastically reduced and the means of interaction narrowed down to adjusting system-wide parameter values. Modelling physiological processes by means of simple, reactive agents that closely follow empirically verified biomedical models allows for an extended interactive exploration and learning experience. Here, the user's interactions are not limited to the parameters of specific classes of agents and system-wide interaction coefficients. Instead, the user can alter individual agents' behavioural configurations, their physical properties, and their states, resulting in vastly heterogeneous models. He may also scale the numbers of interacting agents up and down on-the-fly, assign concrete locations or spatial areas, probabilities of occurrences, and even introduce random variations. Behind the scenes, models could be described at arbitrary levels of abstraction—whether considering biochemical interactions at the molecular level, complex behaviours of cells, or state-based dynamics of organs. In addition, differently represented models wrapped

as agents could be considered in a generic agent-execution loop. As the agent representation makes attributes and states readily available to all model entities, interfacing across multiple scales of time and space would solely pose a logistical but no representational challenge.

Over the years, the Lindsay Virtual Human project yielded several computational training tools which have been utilised to enhance the learning experiences of students of biology and medicine. In addition to the interactive anatomy atlas and tools for authoring and exploring physiological processes presented in Chapter 3, alternative modes of access to the simulated and visualised contents have been introduced. For instance, the means of (remote) collaborative work based on distributed web-services is elaborated on in more detail in Chapter 15. In addition, an augmented reality implementation has been presented that allows to project detailed information on human cadavers to enhance basic anatomy training [122]. Next to the human immune response, which has been modelled and evaluated in rich detail in the context of the Lindsay Virtual Human project [123] and which has served as a basic model for presenting novel modelling techniques of interactive self-organising systems in Chapter 13, other systems have been simulated, such as neural pathways [124] or the inner workings of Escherichia coli bacteria that populate the human intestines [125]. Educational studies on the effect and the requirements for introducing novel interactive teaching media, such as the LINDSAY Presenter, an interactive anatomy atlas, have already been conducted [126]. In general, the utilisation of such kinds of interactive learning aids has found appraisal, but they still represent a very early and still very limited means to computationally enhance medical training [127].

2.1.2 Interactive Dental Medicine

Investigation of means of interaction with medical simulations, especially with respect to physical model representations, is the focus of Chapter 4. The described application domain is endodontic treatment: Therein, infected tissue in dental root canals is removed. The first few steps of the procedure are as follows: (1) A cavity needs to be drilled into the tooth to gain access to the root canal system. (2) Hand instruments are used to explore the root morphology. (3) The access to the root canals is widened by means of a manually operated nickel-titanium

file. (4) A rotary file is then used to extract the infected tissue. After rinsing and cleaning the prepared root canals, they are filled, the access cavity is closed and the tooth's surface reinstated. Especially in the context of steps (2) to (4), the flexibility of the dental instruments has a strong impact on the operation procedure, as it allows the dentist to feel and react to impediments and frictions the instruments are exposed to. In order to approximate the physical behaviour of flexible files, we developed a deformable file model based on articulate body physics.

Due to the focus on interaction and physics, the utilisation of self-organising system models is only hinted at in Chapter 4: Models of self-organisation could drive the simulation of long-term evolutions of treated teeth, especially considering the regrowth of bacterial biofilms that repopulate the root canal system, if the operation was not completely successful, as is the case for 15 to 32% of all endodontic interventions [128]. Another aspect of self-organising system dynamics could address the simulation of recovery of affected tissues. Different kinds of physics, such as heat propagation and dissipation or the propagation of pressure through crystalline dental materials, could complement according self-organising system models.

self-organising models aside, numerous approaches to virtual dental training have been presented. Products like DentSim (DenX Ltd.), video-capture the navigation of instruments by the trainee through LED emitters [129]. Based on the sensory information, objective feedback about the treatment can be provided, thereby ensuring an effective training procedure. Voxel-Man Dental [130] is an exemplary project that combines three-dimensional visualisation and force-feedback input devices. The focus of the interactions lies in the preparation of teeth and treatment of tooth decay. Force-feedback devices transmit forces to the user, promoting a multi-modal, natural, haptics-based communication between machine and user. Voxel-Man Dental differs from its competitors in its effective visualisation routines, its integration into the curricula of dental medicine programmes [130], and the capacity to perform even complex operations such as the amputation of dental roots [131]. Similar systems, also backed by comprehensive education studies, comprise PerioSim (University of Illinois) [132] or the Virtual Reality Dental Training System (Harvard University) [133]. hapTEL (King's College) deserves special attention as it deploys a force-feedback device specifically designed for dental

contexts [134]. The other systems are typically based on the Geomagic Touch force-feedback device (formerly Phantom Omni) [135]. Different from its predecessors, the work presented in Chapter 4 does not rely on force-feedback devices but solely builds on visual, so-called pseudo haptics. Instead of training dexterity, it focusses on the complexity of endodontic treatment and it harbours various opportunities for consideration of self-organising processes during the simulation.

2.1.3 Developmental Biology

The Lindsay Virtual Human project (Chapter 3) and ventures towards interactive dental treatments (Chapter 4) mainly target medical education. But their computer scientific methodology also lends itself well for exploration in biomedical research and ascertainment of parameters of according empirical models. This idea is fleshed out in Chapter 5 which introduces to the field of developmental biological system modelling and simulation. Developmental biologists devise and hone models to make predictions of organismal development at different stages of the organismal life cycle [82]. To this end, they closely examine the properties of single cells, their phases of development as well as their interplay. The developmental processes are self-organising in that there is no central control but the cells concert their interactions and phase transitions based on locally perceived signals and internal states. Cell-produced signals, generically referred to as " morphogens", allow the cells to communicate with their local environment and trigger and inform the rate of proliferation, cell migration, cell differentiation, self-inflicted cell death (apoptosis), the adhesive forces between cells and various other properties and interactions. Due to the interplay of locally confined, neatly timed morphogen emissions, complex morphological processes unfold to ensure the proper supply of the organism at each stage of its development. Under potentially varying environmental conditions and considering interdependencies in the construction sequences, they make certain the required structural and functional infrastructure of tissues and organs are achieved.

The consideration of required, multi-facetted model aspects renders model building and simulation of developmental biology systems a considerable research challenge. These aspects include

physical properties of molecules, fluids, cells and tissues as well as high-level biologically described behaviours of cells and cell clusters. In the context of self-organisation, developmental biology is of great interest due to the highly evolved interaction processes, the arising, observable morphologies and the link between form and function. Although the goals of developmental biologists differ greatly from those of medical students, the need for accessibility to and interaction with the systems' models and simulations is similar—compare the vision of interactive self-organisation in biological laboratories outlined in Section 1.2 and the research efforts of the Lindsay Virtual Human project in Chapter 3.

Considering developmental biological models and simulations, the physical interdependencies and interaction have recently earned enormous credit. In fact, numerous phenomena such as the self-organised formation of supply networks in tissue or the growth and differentiation of plant stems can hardly be retraced in the absence of physical stresses [136, 137, 138]. According, novel computational models of developmental processes incorporate both, biological behaviours and physical properties/interactions. In [139], for instance, cells are modelled as temporarily deformable spherical bodies. After division, daughter cells take up the space of their parent cell and grow until they have reached their full sizes. The cells can establish finely tuned adhesive forces among each other, which can break based on external stresses, and which serve as a basis for forming tissue layers. Generally, it is important that biological developmental models retrace the basic interactions listed in [83]. They include division, induction, adhesion, apoptosis, migration, contraction and matrix modification (swelling, decomposition, or loss). CellSys is another exemplary framework for modelling and simulating developmental processes. It also provides components for visualisation and analysis. Again, each cell is represented as a spherical, elastic body that can divide, grow, and migrate. Deformation, compression and adhesion are implemented following the Johnson-Kendall-Roberts (JKR) model, which defines the contact mechanics between elastic spheres. Explicit Euler integration drives the equations for diffusion across discrete grids and consumption of nutrients and growth factors. Cell parameters that CellSys supports include, for instance, their elasticity, diameter, the diffusion rate of morphogens, surface adhesion, initial orientation of cells.

2.1.4 Architecture

The principles of self-organisation that drive simulations of developmental biology sometimes also guide and inform the design processes of post-modern architectural works [140, 141]. An according computer science course " Biomimetic Computation" was developed and taught to groups of graduate students at the University of Calgary. The course covered the basics of devising and utilising self-organising algorithmic models for applications in art and design. In an accompanying studio course, students from architecture applied their gained knowledge and skills to challenging architectural projects. Chapter 6 presents the corresponding teaching methodology tailored towards interdisciplinary students and shows selected student works. Next to the algorithmic generation of various naturally occurring forms and structures [142], the computer science course addressed the design of computational swarms, the individuals' interactions including techniques of indirect communication through the environment. It also introduced nature-inspired optimisation techniques such as evolutionary algorithms.

Comprehensive programming code examples were provided for the integrated development environment *Processing* [143]. Processing has been developed with an emphasis on fast prototyping of innovative algorithms and their interactive parameterisation. The recently released third version of Processing expands on its previous fast prototyping functionality and provides graphical user interface elements for changing arbitrary encoded parameters during realtime. Under the supervision of lecturer and teaching assistant of the computer science course, the participating student groups were able to sketch out functional prototypes very easily and to explore the outcomes of their self-organising design systems by means of vast parameter studies. Investigated concepts of self-organising generation of form include locally coordinated additive and subtractive construction processes, whereas collisions, pheromone-like triggers, internal states determine the agents' interactions. In the studio course, the students transcribed their findings for architectural design softwares such as Maya and Rhino 3D, which offer comprehensive scripting interfaces. As the last step of the studio projects, the students crafted actual three-dimensional models of their designs to investigate the effects of the created forms and structures under natural contexts, including lighting and considering scales.

Utilising agent-based, self-organising system models for the conceptual design of buildings has grown into an established methodology for complex architectural designs [144, 145, 146, 147]. It provides solutions to the local integration and adaptivity of designs, their functionality, and the ever growing demand for flexibility. The research efforts have not come to a halt at the mere virtual design, however. Instead, great strides have been made in the context of swarm robotics and self-organised construction of building architecture. A termite-inspired robot ensemble of three units, for instance, can solve simple instances of the inverse problem, where the desired globally built three-dimensional artefact is defined by the user, divided into small subtasks, and performed by the robots in a self-organising manner [148]. At the same time, the empirical foundations are being honed. For instance, it has been found that certain termites prefer working on recently deposited construction material rather than on clean soil [149], presumably due to a certain cement pheromone excreted by the individual workers. The importance of pheromones has just recently been stressed again, when it has been found out that in certain ant species their local deposit conveys in rather clear instructions about the growth and form of the resultant construction [150].

2.1.5 Art

Chapter 7 presents several fine arts projects which explore the "liveliness" of self-organising systems. Similarly to the architectural projects in Chapter 6, the interaction dynamics of ensembles and swarms of virtual agents lead to the generation of artificial three-dimensional constructions. Three artists (JW, MW, SvM) first enquire into the arising virtual dynamics, adjust parameters and perspectives in accordance with their interests. Importantly, the underlying interaction dynamics of the agents are captured in three-dimensional geometries as well as in screenshots. These digital artefacts then serve as the inspirational basis of paintings and sculptures.

One artist (SvM) achieves the adjustment and exploration of the simulated dynamics at the same time by means of interactive evolution. Hereby, the artefacts constructed by a small set of randomly initialised agent populations is displayed at first. Next, the artist rates the displayed

phenotypes to increase the probability of the respective *genotypes*' propagation, recombination and mutation in subsequent generations. The interactive evolution loop of simulation-display-rating-evolution is repeated for as long as the artist wants. Both the artists SvM and MW craft pieces that directly relate the virtual agents' behaviours to the interactions of paint particles among each other and with external physical factors. JW complements these explorations with works that bounds the vast variability of the emerging three-dimensional structures, thus making it accessible to the observer in dynamic, interactive sculptural displays. The presented pieces meld traditional media with novel, computer-generated contents in order to drive the artistic discourse on interactive self-organisation. Other media such as arrangements of video projectors in combination with camera-based motion tracking, generative software applications or computer games have frequently served as alternative platforms contributing to the exploration of interactive self-organisation from various angles.

Simulations of interactive self-organising systems have attracted attention from the arts over many years; Therefore, it has been an important, re-occurring theme in the discourse between science and art, most prominently documented by journals such as Digital Creativity (since 1997, Taylor and Francis) and LEONARDO (since 1968, MIT Press). Ever since their programmatic conception in the late 1980s [1], the reactivity of simulated flocks on user input has served in numerous installations featuring motion-tracking and video projections [151, 152, 153]. Reynolds' boids have even been considered an icon of artificial life art [154]. The application of self-organising systems has not been limited to (computational) visual arts but also extended into the fields of sound and music synthesis [155], and choreography [156, 157]. The frequency and the intensity of self-organisation in the arts promises numerous continuing efforts in the exploration of their interplay.

2.1.6 Ecological Exploration

In Chapter 8 the idea of interactive self-organisation in disguise of a computer game is applied to the challenge of keeping an ecological balance in an aquarium. The user may populate a virtual aquarium with snails, water plants and fish. Fish pollute the water with excrements, consume

oxygen and release carbon dioxide. Snails clear the water from dirt but also use up oxygen. Plants, in turn, emit oxygen and rely on carbon dioxide. They, too, contribute to the pollution of the water in parts, due to regular loss of vegetation. Keeping these few interdependencies in check without jeopardising the delicate equilibrium of the aquarium is the goal of the game, while continuously expanding and renewing the aquarium's setup.

The game relies on traditional two-dimensional graphical user interface elements to guide the user and to provide him with clear options for interaction. The basin and the population of the aquarium, its activities and the resultant water quality are at the centre of attention. Along the perimeter of the view, gauges and little diagrams convey exact information to support the merely visual impression of the system state. Input from the user is received only to trigger strategic decisions considering the aquarium population's composition and to inspect the three-dimensional visualisation of the aquarium. Therefore, next to a standard, mouse-based navigational interface, menus from button elements are provided to remove old and add certain new inhabitants.

The serious game presented in Chapter 8 provides a single point of entry to influence the dynamics of the studied self-organising system, namely by changing its constituents. As this limited access to the system corresponds to the possibilities of interaction offered by an actual aquarium, it is not perceived as overtly confining. However, the interaction possibilities offered by countless realtime strategy games, including titles such as Dune, StarCraft, or Battle Zone [158], allow the user to select, navigate, control, and manipulate individual (typically military) units and subsets of the system alike. Here, too, the units would follow default behaviours without the interference by the player. Yet, he may take control to support the self-organising system in reaching its goals. The one-to-many relationship between the player and large numbers of units under his control is even more pronounced in games such as Pikmin and Overlord [159], where the player navigates an avatar from 3rd person perspective, which can, in turn, command his followers. Over the decades, games have introduced a great variety of interfaces for dealing with large numbers of (semi-)autonomous units such as zooming in on individuals, reading the status, reverting back to a global view, selecting units of specific kinds or commandeering hand-picked subsets. For all of these interaction tasks, numerous solutions

have been presented. Some of which worked rather well (such as selection in 2D views), whereas others, such as the introspection of individual units in Carrier Command: Gaea Mission, may seem bothersome. However, systematic research on these and other questions regarding the usability und user experience for handling self-organising systems has just recently begun due to the increasing availability and accessibility of robotic systems (see also Chapter 10). An overview of the current state of the art in human-swarm interaction can be found in [160].

2.1.7 Network Systems

While the relationships among elements are fixed in the ecological exploration game of Chapter 8, the user is empowered with great degrees of freedom for configuring individual elements as well as their connectivity in the serious gaming title on power grids presented in Chapter 9. In detail, power plants (solar, wind, water, nuclear, coal) and consumers are located on a map of Germany. They all represent nodes in a networked system, whereas the links between the nodes can be established and changed by the user. The user is guided step-by-step through the different interaction possibilities and descriptions of the visuals in a comprehensive tutorial level. Next, he can either build and explore the complexities of power grids on his own agenda or he may play through several pre-defined challenges, including, for instance, one level in which a rough approximation of Germany's power grid needs to be altered to be driven by a large percentage of renewable power.

Drag and drop of power plants and consumer nodes, drag and drop to draw new power lines, marking single and multiple network elements by means of rubber-band selection, and deletion and configuration of individual network elements by means of simple two-dimensional buttons and sliders roughly summarise the interaction tasks of the main interface of the title presented in Chapter 9. In the integrated development environment of the game and simulation engine Unity3D, which was used to develop the game, several additional user interfaces are offered for level design. In detail, these additional interfaces allow for regulating the interaction opportunities in specific level scenarios associating permissions of interaction (full access, no deletion, no access) with each interactive element and its attributes.

Having realised the importance of infrastructure networks as the backbone of the engineered society, several interactive simulations have been dedicated to this topic. First released in 1989, the computer game Sim City has a long-time standing in the market [161]. Here, the player learns to layout infrastructure networks while building flourishing cities. Less playful, with a rather tight focus on economics, Power TAC tries to make informed predictions about viable economic settings in a liberalised, decentralised electricity market [162]. With similar goals, the serious game INFRASTRATEGO captured the decision-making strategies from more than three thousand played games against human players that tried to optimise (a) policy making and (b) price negotiations in an open energy market [163]. While these and other titles, directly draw benefit from simulating the networked production and dissemination of power, their common theme is rather universal: Diligently configuring nodes to make them serve the functionality needed in a complex interwoven system, adding missing pieces and slimming down whenever possible—these are the general challenges brought about by the network perspective, whether applied to economics, life sciences or engineering.

2.1.8 Robotics

The work presented in Chapter 10 covers numerous aspects of interactive self-organisation. The goal of the respective project, called OCbotics, is the design of adaptive, autonomous self-organising robotic systems that provide the means for inspection, goal specification and direct interference by their human users. Single and ensembles of quadrotor drones provide the context of the given research. Next to the low-level reactive behaviour of single autonomous drones that weave architectural structures under lab conditions, an elaborate modelling and simulation strategy is outlined to minimise the reality gap, i.e. the ultimately experienced difference between simulated contents and reality. The computational models of drone ensembles are exemplarily utilised for generating and optimising collaborative behaviours for maintenance of building facades. In a way, this architectural application domain augments the virtual experiments presented in Chapter 6 with capabilities to shape reality based on interactive self-organisation on-the-fly.

The aspect of on-the-fly interference with reality renders the need for responsive hardware and appropriate user interfaces an important aspect of OCbotics. In particular, user interfacing is explicitly covered by one of several layers of an observer/controller architecture that is frequently utilised in the field of Organic Computing. The lowest layer is the adaptable system itself (the drone, in the given context). The next layer captures the classification of the situation and, if needed, the adjustment of the system's behaviour. At this second level, a reinforcement component can be deployed as well in order to grow the confidence in individual actions' or behaviours' execution. The third layer realises simulation and optimisation of the system's behaviours over a longer time-span (not realtime). Considering all these aspects of adaptive behavioural logic, the remaining layers on top are tailored towards user interaction and collaboration (with the user but also with other technical units).

The preliminary interfacing options shown in Chapter 10 include the navigation of a drone ensemble using a head-mounted display and 3D input devices (Razor Hydras). The idea is that in an augmented reality setting, the navigator would ease into the control of the ensemble fastest, if the spatial relationships between the drone individuals and himself are naturally experienced, i.e. if the user does not have navigate through the view of an exterior, possibly also mobile camera. Of course, this option still bears great importance as soon as the robots leave the human controller behind. Nevertheless, the transition between the naturally perceived relationships and the leap into an exterior image source can be continuously animated, minimising the potential loss of context, thereby also minimising the translational cognitive load for the user.

OC systems interface with the user to receive initial and adjusted goals and to give the user an opportunity to inspect the system's current efforts and states. Next to specifying goals at a rather high level of abstraction, the interplay of the user with the system, also considering its decentralised, autonomous components, is best faced by means of Human-Swarm Interaction (HSI) methodologies [164, 165]. Besides taking heart in the general user interface design guidelines, which all aim at minimising the cognitive load of the user [20, 21], an HSI interface should support an effective utilisation of the machines' intelligences, promote local interference rather than global ones, and be scalable [166]. Multi-user interaction with the swarm is also desirable,

as is the interaction with subsets of swarm individuals. Ideally, an HSI interface should also be applicable to a multitude of application scenarios and not only one very specific one. An according exemplary interface was already presented in 2006 [96]. Here, robotic ground units coordinate wirelessly, whereas one of them, the gateway robot, communicates with the user and disseminates new instructions to the swarm. The user may also take direct control of individual units of the swarm, thereby influence its dynamics indirectly [160].

2.1.9 Summary: Interactive self-organisation Taxonomy

The chapters of Part II of this thesis present several application domains for interactive self-organisation. Their contents include molecular signalling, physical interactions with crystalline organic structures, interactions of organismal cells, the workings of physiological systems, e.g. the human immune response, the interaction of individuals in ecological niches, and the view on networked system components. The re-occurring theme of self-organisation is utilised in different ways: To stage models of complex systems, to empower robotic units to collaborate, to challenge and train players and students, to let artists explore complex regimes of interactions, to confirm or question research hypotheses of biologists. The modes of access and the degrees of interaction with the respective self-organising systems vary as well. Figure 2.2 provides an overview of a self-organising system's elements/aspects that one can interact with. The user might directly change the overall system state, or the states of individual units of the system. In the figure, this would correspond to moving an individual or all of them, as their locations partially represent their state. The individuals' (internal) attributes are depicted as small purple boxes, if assigned specific values, they are filled with pink. The user might change the number/type of attributes and set them to arbitrary values. Finally, the user might change the interaction topology (depicted as green lines) among the individuals.

In accordance with the accessible aspects of an interactive self-organising system (Figure 2.2), Tables 2.1 to 2.3 categorise the example projects shown throughout the chapters of Part II. Each row references the respective chapter, names the implementation, and conveys information about the accessible elements, the scope of granted manipulation regarding attribute-value pairs

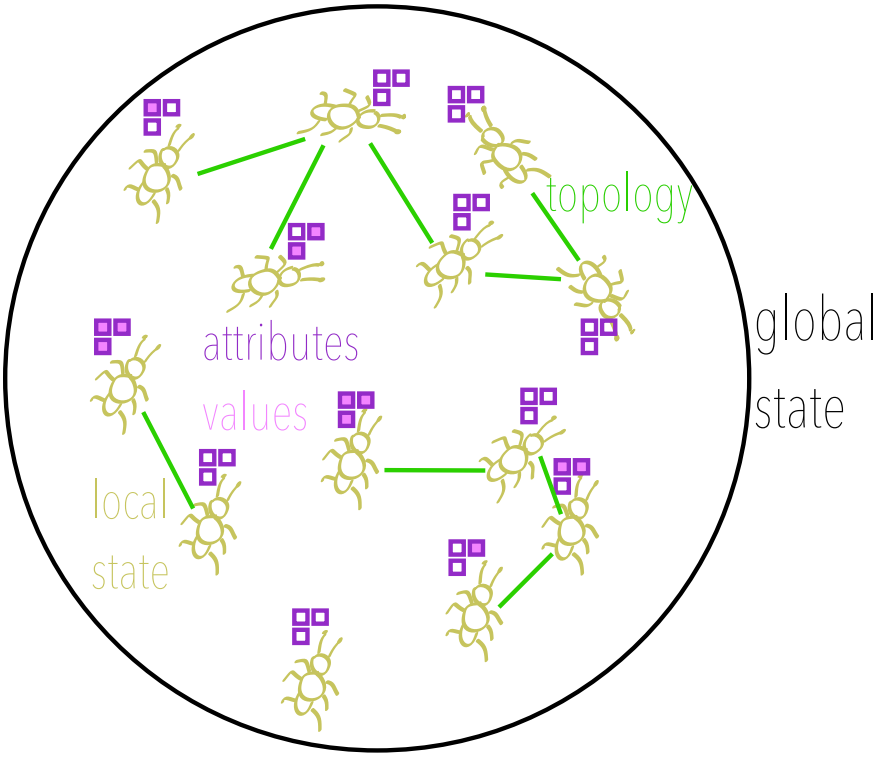


Figure 2.2: Aspects the user can interact with: The global state of the system, the local states of the individuals of a self-organising system (ant schemas), their attributes and values (small boxes), the topology of interaction (green lines).

and interaction topology, and identifies the time frame of access. The multitude of examples allows to establish a preliminary taxonomy; The accessible elements can range from globally perceived scenes, slide sets or models via subsets of data objects, to individual data objects, their attributes, including behavioural rules, the attributes' values, and their states (denoting those attribute-value pairs that affect the elements' relationships to other peers). Selection and alteration of the accessed elements can be performed directly, indirectly by providing selection queries or filters, or at yet another level of indirection, a meta-level so-to-speak, by letting algorithms automatically decide on selection and alteration, as in the case when running optimisation algorithms. The interaction topology may result from the individual elements' states (and internal data values), as well as their behaviours. But it might also be directly set by the user. The time of access can be a rather limiting factor, considering that in many cases, it boils down to a singe period of access, namely before starting the actual simulation. Yet, the means to access model elements during ongoing simulations represent an important factor for modelling, refinement and interactive analysis of self-organising systems. That being

said, to this date, each point of access granted to the user requires the conceptualisation and implementation of a dedicated user interface. The next part of this thesis will shed some light on according interfaces and provide an outlook at how interaction could seamlessly support different times of access and at different levels of selection and detail.

2.2 Overview of Part III: Modelling Aspects

Part III of this habilitation thesis presents works revolving around interactive self-organising modelling methodologies. As stressed in the introduction, this explicitly includes crafting models of interactive self-organising systems but also harnessing interactive self-organisation during the modelling process itself. We cover a broad variety of aspects relevant to model building. We introduce a general modelling methodology and we discuss generic computational representations for arbitrary behavioural relationships in agent-based systems. We transcribe these findings to devise a novel visual programming approach that amalgamates simulation space and behavioural, programmatic logics, and we introduce a symbolic language to intuitively programme locally interacting agents. At the backend, we flexibly organise the agent code by cumulating components, and we organise experiments in databases providing high-level, visual access. Figure 2.3 shows an overview of the chapters of Part III.

First, in Chapter 11, we present the complex systems modelling and simulation approach, a general methodology apt for the development of interactive simulations and especially interactive self-organising systems. It is founded on scientific methods, explicitly considers the empiricist roots of model-building and simulation, emphasises the potential of emergent phenomena and complex regimes in self-organising systems, and provides a constructive, agile workflow. The key concept of self-organisation lies in modelling the interaction behaviours of the involved units. An expressive, concise and easily readable representation is required in order to enable non-programmers to flesh out intricate models. At the same time, the complementing execution algorithm, which is inherently married to the representational data structures in order to provide consistent operational semantics, needs to perform efficiently. On top of these basic re-

Chapter	Project Name	Accessible Elements	Attribute/Values	Topology	Time of Access
3.4	Lindsay Presenter	slide sets, slides, element sets, elements, mesh-regions	spatial relations, visual effects	transitions, slide order	in-depth control during slide composition, high-level control during presentation
3.4.2	Lindsay Presenter - Remote Control	single slides, element sets, elements	spatial relations	slide order	during presentation
3.5	Lindsay Composer	scene, agents, components, attributes	all values	transitions, physics, behaviours	in-depth control during model configuration, high-level control during simulation
4	Endodontics Simulator	tooth, instrument	drilling speed, applied force	manipulation through physics simulation	specific, chosen point of interaction
5.3	SGs Basic Implementation	agent sets	type-specific boiding and reproduction	field of view definition	during model configuration
5.3.2	SGs Decentralised	agent sets and individual agents	boiding and reproduction	field of view definition	during model configuration
5.3.1	SGs EVOLVICA	population, evolution parameters, individuals	individual ratings	implicit through rating	global manipulation during configuration, local ratings during simulation
5.3.2	SGs Breeding Grounds	population, evolution parameters, subsets	spatial relations	evolutionary operators	global manipulation during configuration, subset manipulation during simulation
5.3.3	SGs Rule-based, Evolution	population, evolution parameters	-	-	during configuration
5.3.3	SGs Rule-based, EvoShelf	individual and sets of specimen	play, replay, visualise	storage order	before/after simulation
5.3.4	SGs Swarm Graph Grammars	type-specific rules	rule constituents and relationships	-	before simulation

Table 2.1: Categorisation of the presented projects' interplay between user interaction and self-organisation.

Chapter	Project Name	Accessible Elements	Attribute/Values	Topology	Time of Access
6.3.2	SO Architecture: SO Programming Infrastructure	arbitrary	arbitrary	arbitrary	before simulation
6.4.1	SO Architecture: Sentient Surfaces	vertices (surface agents)	spatial relations	-	before and during simulation
6.4.2	SO Architecture: Diversity Flocks	individual agents	spatial relations, attractive/repelling forces	field of view	configuration before simulation, displacement during simulation
6.4.4	SO Architecture: Procreating Particles	individual agents	spatial relations (over time), procreation rate	collisions through volumes	before simulation
6.5.1	SO Architecture: Decay Swarms	material-specific properties, agent behaviours	spatial relations, boiding, decay rates	collisions through points	before simulation
6.5.2	SO Architecture: Spherical Aggregations	individual agents	spatial relations, merging behaviour	collisions through volumes	before simulation
7.4	Art Studies: Swarm Grammar Collages	agent sets	spatial relations, type-specific boiding and reproduction, visual parameters	field of view	before simulation
7.5	Art Studies: Interactive Sculpture	phenotypes	rating	-	after simulation

Table 2.2: Categorisation of the presented projects' interplay between user interaction and self-organisation.

Chapter	Project Name	Accessible Elements	Attribute/Values	Topology	Time of Access
7.6	Art Studies: Photographs	agent sets and individual agents	spatial relations, type-specific boiding and reproduction, visual parameters	filed of view	before/after simulation
7.7	Art Studies: Acrylic Development	paint	spatial relations, colour, viscosity, heat, gravity, wind	layers	arbitrary
8	Aquarium Simulator	global agent composition	indirectly: metabolic rates	-	arbitrary
9.3.4	Power Grid Simulator - Level Generation	global goals, agents	spatial relations, production/consumption properties, access permissions	interaction topology	before simulation
9.3.3	Power Grid Simulator - Story Mode	agents	accessible properties	direct connection (if permitted)	before/during simulation
10.3	OCbotics - Aerial Construction	agents	spatial relations, flight behaviour	-	before experiment
10.4	OCbotics - Aerial Maintenance	population, evolution parameters, agents	state machine transitions	physical	before/after simulation
10.5	OCbotics - Immsersive Swarm Control	agent sets and individual agents	spatial relations	field of view	arbitrary

Table 2.3: Categorisation of the presented projects' interplay between user interaction and self-organisation.

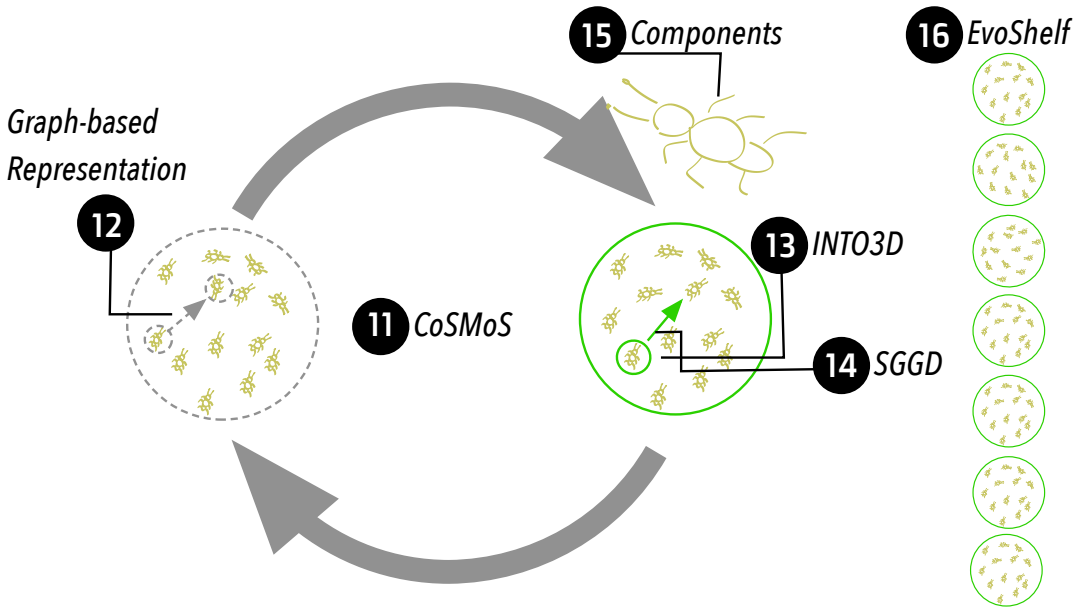


Figure 2.3: The modelling aspects addressed by the Chapters (numbers) of Part III of this thesis.

quirements, it is desirable to establish an axiomatic and thereby extensible vocabulary—again, the representation has to support it. In Chapter 12, we present a rather versatile, graph-based representation for modelling agent-based, self-organising systems. Due to its generality, its performance has not reached the required level but, beyond the aforementioned goals, it allows one to understand the system's evolution as a result of locally executed behavioural interactions. This idea of linking local interactions to the global system state is resumed in Chapter 13, where we present a visual programming/modelling approach for self-organising systems. Previously, simulation space and programmatic editing spaces have been kept separate. The new, interaction-oriented 3D modelling and simulation approach (INTO3D) merges hierarchical graph-based visualisation methods with according, hierarchically and horizontally organised algorithmic execution models. Each agent's behaviour is wrapped in one behavioural module represented as a spherical volume with input and output connectors to related it to its environment. As these modules are recursively defined, one can dive ever deeper into the code of a single agent to finally arrive at the atomic query and action operators that drive the simulation by changing aspects of its state. Although INTO3D allows one to directly link between simulation objects and their behaviours, the 3D graph representation of interaction logics is not as expressive as it could be. Instead of mere nodes and edges, graphical annotations should

express their meaning to facilitate visual programming. This is an idea sought after in Chapter 14. In Figure 2.3, the heading of this chapter is labled with the acronym 'SGGD' which stands for swarm grammars implementation for girls' day. It presents an effort to making programming of swarming agents accessible to high-school students by literally letting them edit their behaviour. On the one hand, one can define which kinds of perceived stimuli trigger an action, on the other hand, the outcome of the action, such as the placement of a construction element in space is specified by visually anticipating this result. More abstract factors such as chance and timed activities are modelled referring to (configurable) symbols such as dice and timers, respectively. Regarding the backend, the multitude of attributes and behaviours an agent can be equipped with renders it the design of an according, flexible and extensible modelling framework a challenge. To this end, we resorted to component-based modelling, which is explained in detail in Chapter 15. Storing and recovering large numbers of experiments represents an essential functionality in the context of modelling interactive self-organising systems. As straight forward as the respective technology can be accessed nowadays, as challenging it is to provide a user-friendly interface for the database backend. This is all the more important as any loss or inaccessibly kept data may jeopardise time and efforts invested in models, simulations and analysis. In Chapter 16, we provide an according, easy-access visual interface to managing large amounts of model and simulation data.

Given this brief introductory overview of Part III, Figure 2.3 can be read as follows. The CoSMoS approach, Chapter 11, is used to describe and guide the development life cycle of interactive self-organising systems (depicted by the centred, cyclic arrows). Thus, it bridges between the real world and the world of modelling. In order to appropriately capture the intricacies of relationships in the real world, a graph-based representation is introduced in Chapter 12 (see the left-hand depiction of a system and the relationships among its agents). Graphical modelling of relationships is applied to a wholly computationally represented model in Chapter 13, which allows for user interference at will (consider the system encircled with green, and the locally drawn, green arrow instructing an ant's activities). The all-visual and interactive perspective of INTO3D can be further enriched by semantic visuals as shown in Chapter 14. Regarding backend technologies, the individuals' properties can be accessed and

managed in an orderly and efficient way utilising components, as introduced in Chapter 15. This aspect is highlighted by the single enlarged ant to the right-hand side of the development life-cycle in Figure 2.3. Finally, the great number of simulations one wants to work with can be handled utilising visualised, integrated database management as presented in Chapter 16 (see the numerous simulation models on the right hand-side of the figure). In the following sections, we will provide a more detailed overview of the individual chapters of Part III of this thesis.

2.2.1 Complex System Modelling and Simulation

In Chapter 11 we elaborate on the adaptation of the complex system modelling and simulation approach, or CoSMoS, to developing interactive simulations. Considering the contents of any simulation, interactive self-organisation can be considered a subset of interactive simulation. However, interactive self-organisation also specifically addresses the means of computational representation, the processes of modelling, of simulation of self-organising systems, of making them accessible, and analysing them. Despite their differences, interactive simulation at large and the field of interactive self-organisation share some goals and numerous challenges. For instance, the need to integrate methods from several rather extensive fields of computer science in order to setup fully functional prototypes. These fields include, for instance, realtime computer graphics, visualisation, human-computer interaction, and realtime physics. In addition, interactive simulations, just like interactive self-organisation, is usually either motivated by retracing complex real world phenomena for training or research, or by anticipating complex novel, engineered situations. Both paths require a systematic, scientific methodology towards conceptualising and implementing model designs. The CoSMoS approach provides detailed guidelines on how to unroll a project in this manner. Its three phases of discovery, development and exploration step-by-step cover aspects such as data gathering about a newly exposed research context, model building, translation to a computational representation, refinement of the computational model as to avoid inclusion of unwanted biases, porting to specific computing platforms and in-depth analysis of the results based on according result models. In Chapter 11, the completion of several interactive simulations is detailed following the CoS-

MoS approach, whereas a special emphasis is placed on the requirements of interactivity of the resultant systems.

In analogy to the summary in Chapter 2.1.9, the interactivity of the systems presented in Chapter 11 mostly accommodate continuous selection and manipulation of aspects of the systems. There are several examples of interactive simulations that entertain models of self-organising systems, for instance simulators about computer network routing, connected cars, beehive defence and immersive experience of ant algorithms, accessible elements. In these simulations, the accessible elements are the individual agents of the system, i.e. servers, cars, bees, and ants, whereas sometimes direct access is granted, e.g. configuration of individual computing machines in the routing simulation, and other times, only indirect access is provided, e.g. influencing ants through manipulation of their physical environment. In all the instances, the behaviours of the interacting units can be changed, e.g. by installing new routing protocols, by tasking bees with goals of foraging or defence, by changing the cars' routes, or by alterating the parameters of perception and action of the ants. Interaction topologies play an equally important role. In the routing and connected cars examples, the user establishes them explicitly. The user guides the bees to broken honeycombs, to infiltrating enemies and to fertile flowers.

The CoSMoS process has been successfully applied to an array of modelling and simulation projects [167]. Its importance to interactive self-organisation lies in its roots of science, its awareness of complexity and self-organisation in system models, e.g. [168, 169]. From these roots grow its attention to scientific detail but also its openness and agility of the development cycle, which is another great asset in light of interactive self-organisation [170]. With the dawn of integrated development environments for interactive simulations such as Unity3D, Unreal Engine, CryEngine, or Stingray, the development of human-in-the-loop systems [67] has received an enormous boost [171]. Next to the accessible integration of methods from different computer scientific fields, great leaps local leaps, such as the introduction of a GPU-supported particle-based physics engine [172] or the coming of age of virtual reality gear and computing infrastructure [173] revolutionise the freedom one enjoys for modelling interactive simulations and especially self-organising systems.

2.2.2 Graph-Based Behavioural Modeling of Agents

Interactions of the involved individuals of a self-organising system represent the core of our modelling endeavours as they determine the global system performance. Accordingly, Chapter 12 is dedicated to the question of how to represent local behaviours. Here, we integrate aspects from formal grammars, graph-based representation and multi-agent simulation in order to arrive at a computational representation that works across scales, and yet, offers an accessible way to tracking transitions from local interactions to global states. In particular, the behaviour of each agent is expressed in rules that substitute local relational graphs, i.e. one situation description is substituted by another. As these substitutions may not only change local states but also introduce new individuals and remove previously existing ones, the according rules are rather generic. The evolution of the system emerges when considering the consecutive substitution of the local graphs across the whole simulation space—a time series of global networks unfolds.

The network perspective of interactive self-organisation, presented in a playful disguise in Chapter 9, is a powerful tool for establishing and resolving interdependencies during the design of a system and for analysis during its simulation and deployment. The expressiveness of networks comes from the fact that relationships among entities are explicitly phrased and are persistently exposed. As a consequence, one could say that graphs and networks provide a perspective that renders a system structure transparent, both to the user and to analytical algorithms. By providing an according means to model local interactions, the user clearly understands the implications of the model behaviour and its meaning for the whole system. He can relate arbitrary agents and their states and translate them into algorithmic instructions to drive the simulation process. This empowerment of the user is essential for interactively designing and altering models of self-organisation.

The idea of representing agents as nodes and edges as inter-agent relations, and of driving their interactions through sub-graph substitutions has also been elucidated in the context of relational growth grammars, or RGGs [174]. RGGs are an extension of parametric L-systems with object-oriented, rule-based, procedural features that result in a universal modelling language, able to

simulate standard L-systems, artificial chemistries and ecological systems alike. RGGs have been shown to grow multi-scale models of plants integrating their structure and function [175] and to grow architectural models [176]. The rationale to describe behavioural changes as local graphs and to understand the global system evolution as a network time series is broadly accepted, nowadays [177]. Initiatives such as those by Kniemeyer and Kurth and Belhaouari et al. to make according graph grammatical formalisms accessible to the domains of architecture and design [178, 179] need to be understood as forerunners of a ubiquitous future of graph-based, network-oriented programming. They are the logical extension of rule-based models of agent behaviours that are currently being established across domains as diverse as biomedical research, economics and social sciences [180, 181, 182, 183].

2.2.3 SwarmScript INTO3D

In Chapter 13, we present a visual programming approach that integrates computational representation of states and behavioural logic and the arising computational processes. In particular, it introduces *SwarmScript INTO3D*, an approach to interactively model and simulate self-organising systems including technically designed systems such as robotic swarms or complex biomedical systems. We have designed SwarmScript INTO3D to allow domain experts without a background in computer science to translate seamlessly between the domain of their expertise and corresponding computational models. Each model entity has to be properly represented, groups of entities be organised into systems and systems of systems, and their interactions concerted horizontally and vertically, across all scales. SwarmScript INTO3D 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 takes the graph-based notion of local models as presented in Chapter 12 and applies it to large system simulations, projecting it right into the simulation space and making it fully interactive for the user. SwarmScript INTO3D 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. Chapter 13 introduces the approach from the user's perspective and sheds light on the execution model and potential abstractions in the context of an interactive self-organisation model of the human immune response.

SwarmScript INTO3D represents the first comprehensive attempt on merging model design and simulation exploration. It does away with the conceptual distinction between design-time and runtime, it nullifies the need to provide specific interfaces for different steps of an interactive self-organising system's life cycle. Rather, it allows the user to fully access agents, behaviours and states at any level of abstraction at any point in time. It further relates local individual behaviours (Chapter 12) and the global system's evolution as well as the behavioural logic, or functionality, to the according system units' visualisations and geometries, or their form. As a result, SwarmScript INTO3D can be seen as a pioneering effort towards unification and simplification of modelling and simulation perspectives.

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 [184, 185]. 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 [186]. Visual programming interfaces are sometimes part of the respective IDEs, as the Blueprint interface for algorithmic modelling working with Unreal Engine [187]or can be added as plugins, as for instance the Antares VIZIO Visual Logic Editor for the game engine Unity3D [188]. In addition to basic arithmetics, and working with abstract data structures, these frameworks provide high-level access to high-level physics calculations and computer graphics functions including 3D asset management and scene graph organisation. Most recently, news have spread

about a virtual reality mode for the Unreal Engine editor [189]. For now, it mainly promises fast 3D scene composition and editing, focusing on basic transformations of 3D objects. However, combined with the trend towards deployment of virtual reality for 3D asset modelling [190, 191, 192], one can infer that the degree of freedom of model creation and manipulation in virtual three-dimensional spaces will continue to grow.

2.2.4 Visual Agent Programming

The previously described chapters span modelling from a conceptional, methodological perspective (Chapter 11), systematic and accessible modelling of individual behaviours (Chapter 12) and the systemic, holistic and integrated view on interactive self-organisation (Chapter 13). Although the latter approach offers drag and drop interaction for building and changing individual behaviours and inter-individual relationships, there is yet another layer of accessibility that can be offered to the modeller/user. This additional layer tailored towards accessibility is introduced by Chapter 14. Here, we present an interactive self-organisation system that empowers high-school students to programme individual agents, to configure them alongside their simulation environment, and to explore their collaborative construction efforts in 3D virtual space. The agents implement functionality to reproduce, to differentiate, and to leave geometrical shapes in space. These activities can be triggered either by peers or certain objects perceived in the environment, by chance, or by timers. The user can model such triggers of perception by simply dragging and dropping the required objects into the agent's projected field of view. The triggered actions can be modelled by dragging and placing the respective agents and objects outside of the acting agent's field of view. Standard graphical user interfaces are offered to adjust any additional parameters, such as the agent's field of view, its flight parameters, its colour or type. Our preliminary trials in the context of an event to promote STEM research among high-school girls shows that the presented approach to visual agent programming is a valuable extension to interactive self-organisation techniques.

Spatial relationships are the underpinning of numerous interdependencies between self-organising agents. Making these accessible alongside the agents' spatial arrangement seems natural.

However, conditions and actions that cannot be spatially expressed need to be considered as well. Therefore, Chapters 12, 13 and 14 suggest to utilise general relationships and let spatial relationships carry meaning whenever naturally understandable. In this way, our spatially trained cognitive abilities can quickly decipher and encode spatially motivated behaviours. In order to provide a similarly quick and clear access to other relationships, the work in Chapter 14 suggests to introduce symbolic visuals whose presence conveys their referred factors' influence in a given situation.

Accessibly programming of agents has received a lot of attention over the years, as programming agents provides a rather direct way of conveying the basic principles of algorithmics [193]. NetLogo is an according, widely spread framework designed to provide easy access to agent-based systems [194]. It deploys a simple but expressive scripting language for describing the agents' properties and behaviours. Visual interfaces for agent-based programming are offered by softwares like StarLogo TNG [195, 71] and SeSAm [196, 197]. StarLogo TNG simplifies programming by wrapping code statements into graphical puzzle pieces—only code snippets that fit together are syntactically allowed. SeSAm, on the other hand, facilitates the composition of multi-agent systems by translating flow charts of the agents' behaviours into optimised program code. Inspired by the first visual programming interfaces for defining LEGO robot behaviours [198, 199], a more generic framework, AgentSheets, was conceived [111]. Here, 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*). 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. The concepts of AgentSheets have grown into a new framework, AgentCubes, which offers students to build complex agent scenes in three-dimensional space [200, 201]. Although AgentSheets can be considered a forerunner in bridging logics and simulation visuals, its creators have yet to introduce modelling relationships directly into the simulation space.

2.2.5 Agent Components

Next to the behavioural logic, an agent typically consists of several other components. These can, for instance, include its visual representation or its physical properties. In Chapter 15 we present a component-based computational framework that allows the utilisation of various formal representations, computation engines and visualisation technologies side by side within a single simulation context. In particular, hierarchies of components can combine attributes and behaviours on different levels of scale and resolution. They can host information for various visualisation techniques e.g., charts and animations, for a range of interaction interfaces, associated with hardware devices or software user-interfaces, or for simulations computed in realtime and driven by heterogeneous computational engines. We apply this framework to design agents composed of graphics, physics and behavioural components which in turn are registered with respective component engines that drive the evolution of the simulation system's state. We showcase the open-ended potential of the component-based approach by introducing a light-weight client/server component, which spreads its siblings in the system's component hierarchy over a network infrastructure. As a result, we can organise, generate, compute and present simulation contents, for instance for medical education, in flexible ways.

In the light of interactive self-organisation, a powerful way to maintain and extend agent capabilities is a strong asset. The breadth of computational methods necessary to establish an interactive self-organisation system, see Chapter 11, demands for the deployment of according software engineering methods, of which, building agents by means of component aggregation is an important one. The benefits lie in the ease of modelling and also the ease of integrating, maintaining, porting and continued development of individual engines.

Component-based frameworks enable human collaborations by disseminating groupware components [202]. Similarly, they can be harnessed to break down and distribute, and reutilise large code bases. One can, for instance, divide the computational work in complex human-computer interfaces into several components for graphical user interface display, for tracking user input and for processing and transforming the input [203]. Interactive simulations and computer

games, whenever multiple processing channels need to work alongside each other, with well-defined interfaces for cross-transfer of few pieces of data, a component-based approach to organising information and algorithms can be beneficial [204, 205]. Component-based approaches afford especially well the concurrent deployment of essentially different calculation engines or engines that need to consider and integrate events at rather distinct time-scales. For this reason, they were, for instance, deployed in the context of an integrative modelling and simulation effort towards various Earth-surface processes [206]. In addition, different from object-oriented programming, the aggregation of components can happen at the front end, not necessarily in code [207, 5]. The flexibility of component-based coding infrastructures is especially suited for networking tasks [208]. Their distributed instalment and exchange of components in data networks can improve the availability and the quality of services [209, 210].

2.2.6 Interfacing with the Modelling & Simulation Backend

Typically, simulations are either parameterised through configuration files or through command line arguments and started from a textual terminal. Scripts can be authored that systematically explore certain parameter spaces, configure and start the corresponding simulation experiments and store them in a semantically structured folder hierarchy. Scripts would again take on the mass interpretation of the experiment results by composing tables and plotting graphs. These visual results can serve as a basis for selective manual analysis by the experimenter, later. In Chapter 16, the last chapter of Part III, we approach the challenge of managing large sets of models and simulation experiments in another way. As before, we bring together several obliquely connected steps in the modelling and simulation processing pipeline, namely scripted data generation, data storage and retrieval and visual data analytics. We accomplish this by providing one integrated simulation and exploration interface, called *EvoShelf*, that provides the user with the functionalities to run scripts to populate a database of models/experiments, to automatically capture screenshots or visualise the experiment results in other ways, e.g. by means of star plots, and to organise and display the staged experiments in an efficient graphical user interfaces.

Interactive self-organisation implies that the user can choose from a large repertoire of choices. Accordingly, the multitude of experiments grows fast. Yet, without a systematic way of organising one's models and experiments, the benefit of interactivity may quickly fade. For this reason, EvoShelf provides an integrated, visual platform that facilitates every aspect of developing, running and analysing simulation experiments. In Chapters 12 to 13, access to individual rules of agents, the rules' constituents, and the arising networks was provided. EvoShelf grants a (visual) handle on whole simulations and sets thereof. Accordingly, it fills a last, important gap to address all major aspects of modelling interactive self-organisation.

Despite the plurality of aspects considered by the solution presented in Chapter 16, the most challenging associated research directions are the scalability of database technologies and methods of visual analytics [211]. Their combined consideration is frequently referred to by the buzzword *big data*, the idea to gather, process and make accessible large volumes of various kinds of data at great velocity and without loss of veracity [212]. As elaborated in [213], according networked solutions rely on fast ethernet connectivity between the database nodes (up to 56 Gigabyte/second in both directions). Although these rates of throughput approximate the ones of local main memory banks, network communication still does not scale well. Physical limitations, for instance introduced by switches, aggravate the situation and prevent the maintenance of steady transfer rates with growing numbers of networked nodes. Data storage, too, is subjected to considerable restrictions. Yet, massively parallel, column-based relational in-memory databases can, in parts, ensure fast analyses of large amounts of data (100TB with clusters of about 100 nodes). In addition, strategies of data partitioning and distribution can drastically accelerate access [214] but the probability of unfavourable data distributions grows with the number of partitions [215, 216]. One idea to embark on this challenge is to partition and place data depending on demand [217, 218]. The same idea needs to be applied when considering the user who needs to quickly sift through the (preprocessed) data in order to understand complex situations or identify anomalies [219]—here, too, data needs to be prioritised or filtered based on its importance [220]. In order to further interactivity of visual inspection [221, 222], an environment like EvoShelf could be integrated into the hierarchical system visualisation approach presented in Chapter 13, thereby making it seamlessly possible to provide

overviews, emphasise important simulations, configurations or behaviours, zoom and filter and provide details on demand [223].

2.2.7 Summary: Modelling, Simulation, Modulation

The chapters of Part III of this work cover several aspects of model-building for and by means of interactive self-organisation. A general methodology for modelling and simulation of complex systems (CoSMoS) is presented in light of interactive simulations. Among the unveiled examples, the contents of several interactive simulations qualify as interactive self-organising systems. They serve as exemplary studies to step through the CoSMoS approach. This approach not only considers a scientifically sound, agile development cycle that supports the design of self-organising systems, but the multitude of aspects from different disciplines of computer science addressed also ensures that it is applied to the development of a rich interactive self-organising system. From this overview perspective of modelling and simulation, we leap right into modelling formalisms for individual agents' behaviours. The conveyed view on locally defined relationships to drive system-wide processes provides an accessible, transparent foundation to modelling interactive self-organisation. The relational, networked view of self-organising systems is taken one step further by projecting programmatic relationships into (three-dimensional) simulation spaces. In order to improve accessibility, still, the means to programme individual agents' behaviours based on spatial relations and symbolically configurable conditions is introduced. Besides local behaviours, other factors may drive the simulation, e.g. physical interdependencies. Also, different modes of access may be deployed as well as various visualisation techniques, or approaches to distributed computation. We show how these seminal extensions of interactive self-organisation can be realised following the component design pattern. The last chapter of Part III wraps up the major modelling aspects of interactive self-organisation providing a solution to integrated design, storage and retrieval and management of large numbers of simulation experiments.

To conclude, Part III draws the path from reality to virtuality in great detail. It also introduces several means to combine previously separated steps in the workflow pipeline of model building,

simulation, analysis and the management of simulation experiments. Altogether, the achieved great level of interactivity of model configurations, at arbitrary level of detail and at arbitrary points in time, even during the simulation process, establishes a new notion of modelling and simulation capacity. In order to give credit to this newly achieved quality of user immersion into the modelling and simulation process, we suggest the utilisation of the term *modulation* which is etymologically close to modelling (from Latin "modulus", the diminutive form of "modus": measure) but stresses the ongoing process of adaptation and, thereby, may refine the notion of simulation (from Latin "simulare": imitate). Rather, as the interactive self-organisation methodology presented in this thesis does not distinguish between the phases of modelling and simulation any longer, but offers an all-encompassing way to model, stimulate, analyse and manage at arbitrary points in time, modulation can be considered the main activity of the user. Hence, the process of modulation begins by laying out and testing the corner stones of a model. It continues when filling it with details, when analysing it from different points of view, when evaluating and refining it. Even when the user interacts with it in an application context, he modulates the model, as he continues to adapt it in accordance to his ideas.

2.3 Overview of Part IV: Optimisation

In the fourth part of this habilitation document, several aspects around optimisation of (interactive) self-organising systems are introduced. Figure 2.4 gives an overview of the most important aspects that we consider. On the left-hand side of the figure, the number of simulated agents is scaled from the bottom to the top. The blue arrows in the figure indicate optimisation flows, i.e. the flow of optimisation that informs or facilitates to calculate other parts of the simulation or modulation. As displayed, we realised scaling agent numbers by looking at and optimising smaller sets of agents and applying the gained knowledge to larger sets. In particular, we accomplish scalability by detecting patterns among certain sets of agents that allow for their subsumption by single meta agents. Thereby, we can reduce the number of calculated agent interdependencies and activities and introduce greater numbers of agents while maintaining low computational costs. At the centre of Figure 2.4, an ant nest construction is seen. Again from

the bottom to the top, increasingly bigger fragments of the construction are shown, starting with a pair of joining corridors. Next, the corridors are attached to a chamber that may, for instance, be used for food storage. From the network of hallways and chambers that arises, the silhouette of an ant hill becomes visible, which is shown in full size and from the outside at the top. The individual steps in the figure emphasise the emergence of greater constructions based on local construction activities and the resultant, locally built artefacts (although the shown steps do not coincide with the actual spatiotemporal succession). The depiction of artefacts is merely motivated by the goal of illustrating the sequential steps in a multi-scale simulation. However, these artefacts result from interaction processes and, hence, their underlying behaviours and properties are the target of any optimisation efforts. The construction elements drawn in golden colour represent artefacts that emerge during the simulation. Their grey reflections, which are slightly shifted to the upper-left, represent the non-computational domain model captured of a real ant nest construction. Optimisation flows inform the simulated model about the observed processes and artefacts at different levels of scale—considering local construction of individuals at ever greater spatiotemporal scales and considering different qualities of the resultant artefacts (corridors, chambers, networks, nest). As the results from the simulated, computational representation may provide the basis for honing and completing the domain model, the respective optimisation flow arrows that bridge between the two models at different scales point in both directions. In addition, vertical bidirectional arrows connect the individual scales of the simulated artefact construction. These optimisation flows indicate the need to (1) ensure reaching higher-level artefacts that closely correspond with the observed domain model based on lower-level interactions/artefacts, and to (2) incorporate behaviours and properties at higher levels that emerge from lower levels of the simulation. Lastly, to the right-hand side of Figure 2.4, we see a schematic display of the development of an architectural building design. Here, the idea is that global goals are defined by the user/designer and the properties and behaviours of the self-organising system are optimised step-by-step to satisfy these goals. Independent of the goal, i.e. retracing empirical models or generating novel processes/artefacts to reach global goals, we refer to the step-wise optimisation and information of emergent processes as guided emergence.

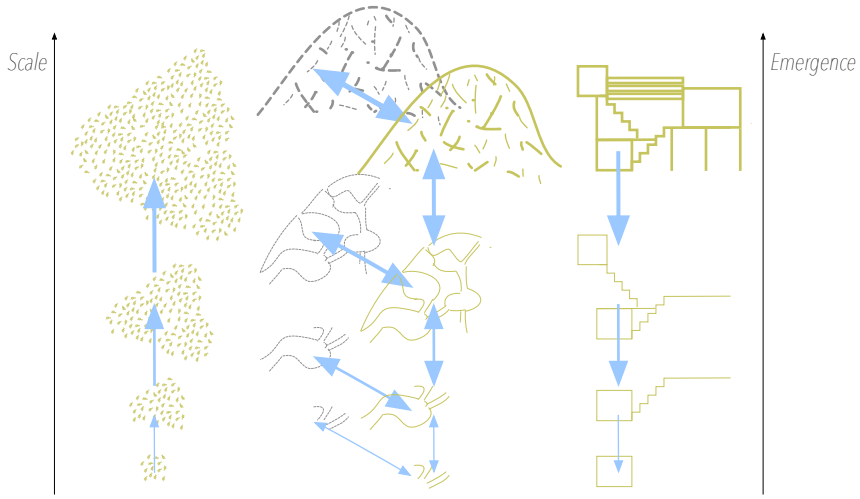


Figure 2.4: Optimisation efforts introduced in the chapters of Part IV of this thesis. From left to right: Scaling numbers of simulated agents, correlating domain models and simulation scales, and top-down guidance of emergent artefacts/processes. The blue arrows represent optimisation flows that inform or facilitate the calculation of other parts of the simulation.

Part IV is divided into four chapters. In the first one, in Chapter 17, we mainly elaborate on approaches towards guided emergence. More specifically, we shed light on top-down model optimisation for (a) self-organised flight across pre-defined two-dimensional surfaces, (b) swarm flocking in three dimensions that approximates a pre-defined progression of flock densities, (c) the guidance of the self-organising construction of three-dimensional artefacts based on pre-defined shapes, and (d) the approximation of a differential equation model of predator-prey systems by an agent-based, self-organised model. Chapter 17 concludes with an outlook on scaling simulation processes through dynamic model adaptation, which is introduced in detail in Chapters 18 and 19. In the first, we focus on speeding up a compartmentalised generational model of the Mitogen-activated protein kinase pathway, that plays an important role for cells receiving and reacting to external stimuli. Here, we identify correlations among the compartments, or agents, and approximate the behaviours of interdependent clusters using artificial neural networks and genetic programming. The learned models allow us to introduce simplified rule sets to drive the calculations. In the latter chapter, Chapter 19, we focus on the idea of immersing model-adapting agents into a simulation. These agents would automatically observe interacting agents from the domain model and subsume them by meta agents whenever possible. To this end, we stress the formalisation of the involved agents, considering their behavioural rules (Chapter 12) and their other properties (Chapter 15) such as their physical

shapes. Next, we introduce an algorithmic loop for logging, learning & abstraction and validation to build and, if necessary, revoke hierarchies of abstracting meta agents. We demonstrate the feasibility of the chosen approach in the context of a spatial, three-dimensional, biophysical simulation of blood coagulation in the human body. In the last chapter of Part IV, Chapter 20, we draw the big picture of automated abstraction in self-organising systems. We elaborate more in-depth about the opportunities provided by the concept of self-organised middle-out abstraction, i.e. the concept to build, maintain and revoke abstraction hierarchies to always consider the required level of detail of a self-organising simulation. We clarify the relationship between abstraction hierarchies for performance optimisation and abstraction hierarchies to pinpoint and express emergent processes and structures. And we conclude the chapter with an outlook on the application potential of this approach, not only considering simulation efficiency and the integration of large, multi-scale models, but also to fill gaps in empirically ascertained models, to guide empirical research studies and to bring the concept to good use by introducing the underlying algorithms to according smart hardware networks. In the following sections, we provide more details about the individual chapters. We emphasise their contributions to interactive self-organisation, and we touch upon respective related works.

2.3.1 Targets of Optimisation

In Chapter 17, we focus on approaches to guide emergence in self-organising systems. We present four according approaches and conclude with concepts regarding the optimisation of the simulation process itself. In the first approach, we evolve boids, i.e. spatially interacting agents that organise their flight patterns based on local neighbourhood perception, to flock in the predefined areas. The boid agents move about in a bounded, two-dimensional simulation space. The user can place tiles beneath the agents to guide their behaviour—the performance of the flock results from the accumulated time of individual boids on top of a tile. In one experiment, after placing tiles in a pattern resembling the holes of a pool billiard table, a flock evolves that breaks up into several clusters to reach the corners of the simulation space. In another experiment, wherein a broad strip of tiles is laid out asynchronously to one the left of

the simulation space, one flock with great performance values is bred whose spatial dimensions resemble those of the tile strip. Another specimen achieves even greater performance by splitting and then re-joining in order to gather as many flock members at the respective spatial position to optimally traverse the tiled area. Different from two-dimensional experiments that focus on the flocking behaviour itself, the next set of simulation experiments targets the parametric optimisation of swarm agents to construct three-dimensional artefacts. To this end, we again use evolutionary algorithms to breed swarms guided by geometrical constraints. This time, the fitness of a swarm is determined by the ratio of building blocks built inside and outside of a predefined three-dimensional structure composed of a set of cubes (a natural extension of the tiles in two-dimensional space). We deploy differently shaped pre-defined guiding structures to push the swarm building upwards or to constructing stair-like, bent structures. We also entrain a constructive swarm incrementally, first breeding a swarm that builds upwards and introducing a bent extension to the guiding structure to instigate optimisation runs staged on the previous ones. Next, we consider the approximation of goals that change over time. To start with, we optimise the boid parameters in such a way that the resultant flock's density over time coincides with a step function and a sine function. Here, the fitness of the bred swarms is inverse to the deviation from the goal integrated over time. Similarly, as a test bed for learning the behavioural parameters of heterogeneous swarms, we choose a classic predator-prey model, in which the populations of prey and predator individuals depend on one another. Sets of simple differential equations describe the dynamics of the according system [224, 225]. We retraced these dynamics, that spring from empiric observations, by means of an agent-based, self-organising model, where predator and prey agents wander about a two-dimensional plane. Prey reproduces, dies on contact with a predator, or dies of other causes. A successful hunt prompts the reproduction of predators, that only die of natural causes. Again, deviation from the predefined functions determined the fitness values of a system configuration. Beside the application scenario and the generation of population target values, the only other difference to the preceding approach lies in the use of particle swarm optimisation [226] instead of evolutionary algorithms.

Different from the methods introduced in Part III, that mainly aimed at bottom-up design,

refinement and modulation, Chapter 17 mainly addresses the challenge of high-level goal definition for self-organising systems. In case of the two-dimensional tiling model, a concrete user interface is provided to specify a desirable target. In the other cases, i.e. the three-dimensional guiding structures, the preset mathematical functions, and the differential equation system of the predator-prey model, user interfaces can only be conceptually inferred from the respective target representations. In addition to these representations, which impose multi-dimensional static or dynamic constraints—one could also speak of corridors of boundaries within which the systems are tailored to work, the contents of this chapter are mainly geared towards population-based configurational optimisation. The link to interactive self-organisation is two-fold. On the one hand, as shown in the tiles example, an interactive application may actually offer interfaces for the optimisation of self-organising systems, possibly even at realtime speeds. On the other hand, the manual introduction of high-level goals is also always a defining element in interactive self-organising systems, as it determines the agents' autonomous behaviours.

Setting and achieving the emergence of high-level goals of self-organising systems has been proposed for various application domains. For instance, by following a hierarchically concerted divide and conquer strategy to reliably and economically meet the production goals in large heterogeneous energy grids [227]. A survey on software engineering methods specifically for programming self-organising systems can be found in [228]. The domain of spatially coordinated self-organisation, more specifically of self-organised construction has been having a fundamental impact on the field of swarm intelligence and self-organisation at large [229, 230]. Until today, self-organising construction is a prospering field of research, see, e.g. [148, 231, 232]. It has been broadly acknowledged that guiding self-organisation is an important concept to mastering complex systems. An overview of *guided self-organisation* (GOS) can be gained by reading [233]. This scientific field notably utilises formal methods of information theory, graph theory and nonlinear, complex systems to characterise and influence self-organising systems [234]. In particular, GSO is generally expected to yield an increase in organisational structure or functionality without providing explicit instructions. Recent research successes are, for instance, the formulation of a novel measure to quantify sensitivity gains of parameters dependent on state transitions [235]. Another measure, referred to as empowerment, captures the degrees of

freedom of an agent to prepare itself for different eventualities [236]. It includes both the means of control over the environment and the awareness of this control. Consequently, empowerment can be understood as a measure to describe the impact of an agent on a system, and, as such, also as foundational information to guide a self-organising process. One more way to finding out about local influences lies in studying the flow of information in self-organising systems. It allows us to identify and quantify causal effects based on probabilistic inference, as seen, for instance, in the example of the sensorimotor loop of mobile agents [237]. A more direct attempt at governing complex, self-organising systems is found in generating functionals [238]. It builds on the insight that one may only want to provide fuzzy goals rather than a concisely specified target state of a system and on ways of optimising according probability functions. A compromise between bottom-up and top-down specification is realised by interactive evolutionary computation methods, that are, for instance, used to manually breed swarm chemistries, where differently configured, proactively moving particles bond with and repel each other [239]. Interactive evolutionary computation translates the user's subjective preferences into fitness values that guide the heuristic search of evolutionary algorithms [240].

2.3.2 Abstraction of Interaction Processes

After presenting several means to optimise self-organising systems to better reach predefined goals or approximate empirically observed phenomena, Chapter 18 is the first one fully dedicated to the concept of automated model abstraction and simplification at runtime. The main goals are to speed up the simulation by simplification and to enrich the model and the results by abstraction. As especially non-scalability [241, 242, 243, 244] holds back agent-based simulations of self-organising systems from finding wide-spread use in scientific research and application-oriented development, we aim at dynamically abstracting agent behaviours at runtime and to thereby reducing the computational effort. Patterns of interactions and patterns of states that are discovered among model agents during runtime allow to recursively rephrase and simplify the agents' definitions and behaviours. The concrete model that serves to exemplify our methods is the Mitogen-activated protein kinase, or MAPK, pathway. It is a signalling

pathway that describes how information travels from the receptors of a cell to an inside target by means of enzymatic biochemical reaction cascades [245]. It plays a key role in the cell cycle and triggers responses to extracellular stimuli and regulates cellular activities, such as gene expression, mitosis, and differentiation [246]. There are two viable established models of the MAPK signalling pathway. One assumes that a stimulating enzyme triggers the production of the information carrying enzyme inside a cell [7]. The other one deploys a negative feedback loop to cause a sustained oscillations in the information-carrying enzyme production [8]. Both models are represented as sets of differential equations. For our model, we consider each substrate a loosely defined, independent agent. We cluster and abstract the function of gene expression by identifying correlations of the respective rates. For learning effective clusters, we utilise the back-propagation algorithm on feed-forward artificial neural networks (ANNs) [9] as well as genetic programming [247]. Our results show the effectiveness of automated hierarchical, observer-driven abstraction.

From the perspective of interactive self-organisation, the presented automated abstraction at runtime promises the accomplishment of two important goals. Interactive systems need to be responsive to the extent that the user can immediately recognise the impact of his interference. In agent-based systems, however, especially if the agents have the ability to reproduce, the computing hardware becomes a bottleneck as soon as too many variables and interdependencies are considered. At this point in time, the computation is staled, the system becomes unresponsive, and usability and user experience fall apart. One can introduce drastic limitations such as a maximal limit of agents in the simulation but this might heavily impact the simulation dynamics. Automatic model abstraction at runtime presents an alternative. In the long run, it might enable the system to calculate all interdependencies across multiple scales without loss of speed or accuracy and yet allow the user to direct the computational efforts to wherever his interests lie. The user-directed focus could impose a more detailed level of simulation and visualisation [248], as if investing the details of interaction on an ant hill by means of a magnifying glass. In addition, automated model abstraction during runtime offers a way to make higher-level phenomena accessible to the user by identifying and abstracting the links between local interactions and higher-level emergent phenomena. Especially in large,

bottom-up systems this task will become increasingly important in order to render them useful. Otherwise, one might be overwhelmed by the sheer dimensions and numbers of interacting units. Lastly, automated model abstraction during runtime also aligns well with the goals of the previous chapter, Chapter 17. In order to understand and evaluate the reached state in an interactive self-organising system, it is seminal to describe this state at a higher level.

Clustering is the starting point for our approach to creating meta-agents that represent groups of domain agents, thereby reducing the computational load of the model. Especially derivatives of the online divisive-agglomerative clustering approach, or ODAC, are suitable for this task [249]. They adjust cluster affiliations incrementally and dynamically over time. Those agents could be considered part of a cluster whose members frequently interact with each other or whose states and actions correlate mathematically. In the long run, the latter could be determined relying on dependency measures such as mutual information [250, 251]. This would be one way to neutralise the bias towards spatial relationships of our automated abstraction approach and to extend it towards arbitrary domains. As an alternative to re-organising our models based on explicit measurements of emergence, artificial neural networks can take on this task. They have been shown to effectively extract features relevant for retracing diverse sets of models, e.g. [252, 253, 254], and they can build and process data in a hierarchical fashion. Auto-associator networks (AAs) represent a special type of artificial neural network that trains a weight matrix to reconstruct the input [255]. As such, they can be utilised for re-building incomplete data or to filter noisy data. Since the mid 2000s, the significance of AAs has surged as the so-called contrastive divergence algorithm was applied to train several layers of hidden nodes in AAs one after the other [256, 257]. These multi-layered AAs, or stacked auto-associators (SAAs), yielded unprecedented performances. Especially SAAs and deep belief networks (DBN), that learn probabilistic estimates in a hierarchical fashion, have been driving the popularity of deep learning approaches [258].

2.3.3 Immersive Abstraction Agents

Chapter 19 extends the preceding chapter by proposing another type of abstraction that aims to build adaptive hierarchies of spatial agents during the course of the simulations. In order to provide a rather flexible means to automated model abstraction at runtime, we follow the idea to devise observer agents that are immersed into a simulation. The observer agents monitor groups of domain agents. Among those observed agents, cluster of agents may be identified based on interaction patterns or state correlations. These clusters are the basis for the creation of a meta-agent that subsumes the domain agents. As before, these meta-agents might not persist, if the conditions for their formation is not warranted any longer. Hence, the observer needs to break down affected meta-agents into their constituent individual agents. In order to determine the persisted validity of a meta-agent, we regularly inspect it. If it can be maintained over a long enough time, it itself can get subsumed by even higher-level meta-agents. In order to substantiate our approach and to provide a basis for its rigorous analysis, we build it on a formal definition that also considers the agents' component-based architecture that, a.o. defines its biological and physical properties and behaviours (see also Chapter 15). Great emphasis is placed on the timing analysis used to attest the benefit of the abstraction mechanism in improving the runtime of a simulation. Here, we focus on the engine performing biological behaviour as it represents the major bottleneck in our system (graphics and physics are greatly optimised by default). In particular, we consider all behavioural components registered with this engine and reveal that the computational complexity of the simulation mainly results from the number of agents and the number of interaction rules of each agent. In order to reduce the number of domain agents, the immersed observer agents continuously perform the aforementioned triple of logging, learning & abstracting, and validating. In case cluster identification and observation trigger an abstraction, the newly created meta-agent's behaviour component aggregates all the unique rules of the subsumed agents. The subsumed physics components are merged into a higher-level physics composite structure and the transformation of the higher-level agent is calculated as the average of all the subsumed agents. During the validation phase, the predicted activity of the subsumed agents is compared to their actual interactions as they are temporarily

fully reinstated. If prediction and actual performance diverge, the abstraction is revoked and the domain agents released again. Otherwise, the confidence in the prediction is increased, which leads to a reduced frequency of proactive validation phases. In our empirical studies we show that our approach increases the performance of a three-dimensional biophysical blood coagulation simulation by 20%. This result considers the computational overhead caused by our approach and the achievement of the same simulation accuracy as without the abstraction mechanisms in place.

The extensions and innovations of Chapter 19 bring about new opportunities for interactive self-organisation. Primarily, of course, the approach's versatility stands out: By merely introducing observer/abstraction agents into an arbitrary simulation (which offers an according software interface), we can achieve optimisation during runtime. Even more, once the agents are trained for a particular simulation environment, they may be recycled and deployed in additional experiments even before a simulation begins. In this case, the agents can pre-optimise the simulation without loosing their ability to adjust themselves and the simulated domain model to new developments and requirements during runtime. Next to versatility, the agent-based approach to automated model abstraction at runtime facilitates a transparent, potentially self-explaining presentation of ongoing abstractions. In analogy to the outlined analysis of computational complexity introduced by domain agents, the immersed abstraction agents can exhibit, each for its own or in ensembles, the computational resources they occupy. The agent representation of the optimisation process also gives the user not only the opportunity for its fine-grained inspection but also to exercise control over it. The user can, proactively, configure the abstraction agents, e.g. exclude their efforts from certain aspects of a simulation, modify abstraction hierarchies, increase or decrease locally afforded accuracy values, direct abstraction agents towards hot spots in the simulation that consume overly much computational power. The user may even trigger the proliferation of abstraction agents in order to ensure coverage and saturation of the simulation space. With respect to the semantic abstraction hierarchies that might be built, the agent-based approach provides a wrapper for correlating states and interactions in a system and their higher-level meanings. Chapters 12 and especially 13 provide according interactive self-organisation techniques modulation of models, optimisation and abstraction.

Merging numerous agents into single meta-agents can simply be achieved by adding an extra variable that denotes how many domain agents it represents [259]. One can find domain agents that can be merged by conducting a principal component analysis of linear combinations of the their attributes [260]. The resultant clusters are further subdivided to reduce intra-cluster variation and then subsumed. Instead of considering a fixed, permanent relationship between meta-agents and domain agents, so-called compression manager agents may direct the formation of meta-agents and manage their integration into the model. Subsumed domain agents may also be released again, if their monitored behaviours exhibit too great variance from their assignee. Mere aggregation of agents renders the model more compact, possibly simplifying it due to omission of detail. However, if the subsumed agents' interplay resulted in emergent properties or behaviours, like hypercycles in chemical reactions, the new representation can be considered a helpful bottom-up abstraction labelling a new structural and functional entity [261]. In artificial chemistry simulations, the emergence of increasing complexity levels can be traced [10, 262]. Relying on rich functional descriptions of genes and molecular assemblies, according hierarchical representations have also been used to analyse and improve the metal-leaching efficiency of certain bacterial colonies [263]. In case no multi-level semantics can be provided upfront but high-level properties emerge from subsystem interactions during a simulation, they need to be detected algorithmically [264]. The need for automatic detection of emergence has given rise to emergence measures considering (discrete) differences in entropy regarding attribute values of a system [265] or their divergence [266]. In the latter case, similarity measures for probability distributions, such as the Hellinger distance, have been used to calculate the divergence over a given period of time. As this calculation is rather expensive, it has been proposed to approximate one distribution by means of a Gaussian Mixture Model and make estimates fitting individual sample points of the other distribution [267].

2.3.4 Self-Organised Middle-Out Abstraction

In Chapter 20, originally published as a chapter of the book *The Computer After Me* [121], represents our most comprehensive view on the subject of optimisation, emergence, and abstraction

to date. It re-iterates our work preceding works in this field, starting with the early concept of self-organised middle-out abstraction (SOMO) from 2011 [268], which basically fleshes out all the general ideas presented and evaluated in Chapters 18 and 19. As a consequence, these results are briefly summarised again. The chapter motivates our approach, i.e. self-organising middle-out abstraction, with the need for integrating large, multi-scale system representations [87, 269], for abstraction to maintain computability, and for capturing emergence [270]. Therefore, we elaborate on the concept's application to simulation systems that exhibit emergent processes and structures, as for instance shown in the domain of artificial chemistry. Here, it was shown before that molecules form recursively that exhibit vastly different behaviours and properties than their constituents. In this context, we detail the technical foundations of SOMO, including deliberations about the deployed data structures to represent agents and their interactions, about learning patterns working with concrete data sets, about creating and adjusting abstraction hierarchies down to the operational level, about the calculation of confidence measures for proven abstractions, and even about the execution loop of the algorithm. We also draw the analogy between agent hierarchies and horizontal interactions among the nodes of these hierarchies to the graph-based visual modulation approach presented in Chapter 13. On top of the concepts presented in Chapters 18 and 19, we further the notion of a self-organising abstraction approach by detailing means of propagation of abstractions among SOMO agents, for instance by gossiping as well as their goal-driven proliferation and optimisation, e.g. relying on evolutionary algorithms. Finally, we deliberate about projecting SOMO technology onto distributed real-world computing systems such as smart sensor networks and how it could quickly build hierarchical empirically investigated models of arbitrary real-world situations in a self-organising manner.

The contributions of Chapter 20 to interactive self-organisation are mainly of conceptual nature. However, these concepts are quite powerful. In particular, these include the advancement of SOMO to fully self-organise to comprehensively cover and abstract vast simulation spaces. The evolutionary processes that drive its self-organisation may well be the foundation for versatile collections of SOMO agent configurations apt for specific simulation and modulation needs. Furthermore, Chapter 20 explicitly bridges the gaps between building abstraction hierarchies

and simulated emergence as well as between the distributed abstraction process and its application to networked hardware. More specifically, SOMO could operate on top of arbitrary self-organising technical systems such as smart sensory networks, whereas the nodes are augmented with effectors to make self-directed inquiry possible, nourish each others' efforts of data accumulation, abstraction and fusion [271, 272]. Based on the gathered data, the nodes would build abstraction hierarchies not only to better grasp higher-level concepts in their environment but also to understand the important relationships between the structures and processes around them. Eventually, such SOMO-driven hardware networks could be used for scouting and intelligence gathering, for self-organised surveillance and anomaly detection. If a user introduced local, meso, or global goals, the SOMO-built abstraction hierarchies would become the backbone of evaluating and honing system and environment.

SOMO harnesses hierarchical organisation of agents to reflect abstraction and to introduce model simplifications. Organising multi-agent systems in hierarchical fashions is rather broadly established and may, for instance, reflect prioritisation or a hierarchy in command [273, 274]. While SOMO builds up hierarchies to backup a model, or rather a simulation, as it mainly considers patterns in the flow of data, the resultant use of the hierarchical data structure should better be considered holonic [275]. Holons describe not only the hierarchical, recursive structures that frequently occur in nature, featuring stable subsystems at different levels of scale, but they also consider their horizontal interconnection. In SOMO and SwarmScript INTO3D (Chapter 13), these interconnections are the relationships, their interactions and their resultant state changes that drive a simulation. Holons represent a powerful concept that can be harnessed for engineering complex systems, e.g. by introducing sets of holons that play different roles for adaptive production control [276]. Yet, unsupervised learning in order to pervade system processes typically happens in hierarchical ways. Artificial chemistries are a prominent domain for learning hierarchies [277, 261]. The formation of high-level molecular micelle structures from polymers from monomers is one of the early examples to show hierarchical emergence based on simple forces of attraction and separation among particles [10]. It has been suggested that automatically detecting emergent phenomena poses a greater challenge [264]. Similar to the SOMO concept, other systems have been proposed to identify and subsume emergent

phenomena in multi-agent systems [278, 279]. Once identified, the SOMO concept envisions the local utilisation of an abstraction but also its dissemination to other abstraction agents. Gossiping is an exemplary, robust method to realise the required communication paths in a networked multi-agent system [280, 281].

Part II

Application Perspectives for Interactive Self-Organisation

Chapter 3

LINDSAY Virtual Human: Multi-scale, Agent-based, and Interactive

We are developing *LINDSAY Virtual Human*, a 3-dimensional, interactive computer model of male and female anatomy and physiology. *LINDSAY* is designed to be used for medical education. One key characteristic of *LINDSAY* is the integration of computational models across a range of spatial and temporal scales. We simulate physiological processes in an integrative fashion: from the body level to the level of organs, tissues, cells, and sub-cellular structures. For use in the classroom, we have built *LINDSAY Presenter*, a 3D slide-based visualization and exploration environment that presents different scenarios within the simulated human body. We are developing *LINDSAY Composer* to create complex scenes for demonstration, exploration and investigation of physiological scenarios. At *LINDSAY Composer*'s core is a graphical programming environment, which facilitates the composition of complex, interactive educational modules around the human body.

Christian Jacob, **Sebastian von Mammen**, Timothy Davison, Abbas Sarraf-Shirazi, Vladimir Sarpe, Afshin Esmaeili, David Phillips, Iman Yazdanbod, Scott Novakowski, Scott Steil, Carey Gingras, Heather Jamniczky, Benetikt Hallgrímsson, and Bruce Wright: *Lindsay virtual human: Multi-scale, agent-based, and interactive.* In: *Advances in Intelligent Modelling and Simulation: Artificial Intelligence-based Models and Techniques in Scalable Computing*, vol. 422, pp. 327–349, Springer Verlag, 2012.

3.1 Motivation

Health care systems all over the world are struggling to provide affordable and comprehensive care. Consequently, the need for excellent education and training of medical staff and doctors is more important than ever. At the same time, emerging digital technologies render it possible to present complex contents faster and better to a wide range of audiences. Within this educational context, we are developing *LINDSAY Virtual Human*, a project at the intersection of Computer Science, Education, and Medicine. *LINDSAY* provides a collection of computational tools for research, presentation and learning in the context of human anatomy and physiology. *LINDSAY* also serves as a testbed for interactive computer graphics, touch-based user interfaces, multi-scale modelling, and scalable computing solutions to run simulations seamlessly on a diverse range of computing platforms: from web-based systems and desktop computers to laptops and mobile devices.

3.1.1 Starting with Virtual Anatomy

As a starting point for this project, we attended and analyzed lectures on human anatomy given by senior instructors from the Faculty of Medicine at the University of Calgary. One of our conclusions was that a virtual 3-dimensional human body in combination with the ability to easily navigate and label the displayed contents could greatly facilitate and improve the learning experience for students. Consequently, such an application for interactive 3D content presentation became the first milestone of our *LINDSAY* project.

3.1.2 Bringing Virtual Physiology to Life

LINDSAY Presenter supports the visualization and exploration of static contents on human anatomical structures. Integrating human physiological processes into an anatomy model is challenging. Highly dynamic processes—resulting from a densely connected network of interacting components—have to be illustrated over time and within 3-dimensional spaces. We did not want to resort to mere animations of such contents [282], as this would not allow for a wide range of interactive exploration of the underlying networked units and their interactions.

Furthermore, we want users of *LINDSAY* to be able to navigate to any part of the virtual human body and, even more importantly, be able to quickly zoom to physiological processes at (almost) any scale. More precisely, a user should be able to experience human anatomy on the whole-body level, on the level of organs, tissues, and cells, down to sub-cellular components. This requires an integration of mathematical and computational models of the various processes across multiple scales in space and time. *LINDSAY Composer* is our first prototype of such a multi-scale programming, composition, and visualization toolset. It allows us to explore and present physiological processes within a virtual 3D female and male anatomy model.

3.2 Related Work

Before we expand on our approach to building a virtual human, we give a brief overview of virtual anatomy, the use of anatomy atlas databases, and the use of component-based frameworks for building dynamic software environments.

3.2.1 Replicating Human Anatomy and Physiology

For more than two decades, scientists have been exploring ways to enhance medical research and education through computer-based renderings of human anatomy and physiology. The American National Library of Medicine started as early as 1989 with the composition of a comprehensive imagery database of human physiology, also referred to as the Visible Human [283]. Since then, these data sets have inspired a large number of virtual anatomy projects for research, patient consultation and education [284].

3.2.2 Virtual Human Anatomy and Physiology

In the context of education, atlases have been composed that promote the exploration of detailed anatomical terminology in a proper visual context [285]. Some systems have been further extended to incorporate data about actual biomedical processes in the human body, for instance processes of gene control [286]. One of the projects at the forefront of modeling physiology by integrating processes across scales is the Physiome project [287], which takes signalling

and metabolic pathways into account to connect protein models with cell simulations. The objective is to build tissues and organs through this bottom-up approach. Such systems can be modelled and simulated by means of traditional mathematical methodologies or as large sets of self-organizing bio-agents, or *swarm* systems [288, 289, 290]. The latter, agent-based approach forms the foundation for our computational models to implement physiological processes across multiple scales.

3.2.3 Components as Dynamic Building Blocks

In addition to multi-scale modelling, we rely on a component-based architecture for a framework to generate and deliver anatomical and physiological contents. The required diversity of data types and computational processes for an integrative simulation and presentation tool can be addressed by a component-based software architecture [291]. More specifically, a component can be broadly defined as a unit of independent deployment, which has a persistent state [292]. The design of component-based software architectures has advantages in regard to various application domains. Frameworks for (human) collaborative work can be implemented by brokering groupware components [202]. The coordinated execution of heterogeneous components works for organizing human collaboration as well as complex code bases that comprise large sets of interoperable, reusable software components. For example, a component-based augmented reality framework could manage elements and modules for user interfaces, tracking or object modelling [203].

Computer games face a similar challenge of integrating vast numbers of software components, whether related to contents, providing networking infrastructure or user interfaces [204]. Component-based frameworks have been investigated and applied in the context of Massively Multiplayer Online games [205]. For practical reasons, multi-facetted game units are defined by aggregating distinct software components rather than by using established methodologies of object-oriented inheritance [207, 5].

An overview of component-based client/server frameworks is provided in [208]. Sets of component-based distributed embedded systems facilitate coordinated interactions [293]. Redeployment of components across a network infrastructure results in improvements regarding service availability [209]. Alternatively, mobile devices can exchange software components in peer-to-peer

networks in order to address user requests [210]. The exchange of software components in heterogenous hardware infrastructures might require adaptation of the control over the respective components or of their data. In [294], such an adaptation strategy is presented for transferring components among devices of varying degrees of computational power, e.g., from a desktop/server to a set of mobile devices. In particular, adaptation is realized by specific drivers that serve as middleware to translate the broadcast software components.

3.3 The LINDSAY Virtual Human

We started developing *LINDSAY Virtual Human* in May 2009 as a collaboration between the Department of Computer Science in the Faculty of Science and the Undergraduate Medical Education program in the Faculty of Medicine. The *LINDSAY* project currently has a demonstration room that replicates a medium size class room (20-30 students) with a large rear-projection screen. With *LINDSAY* we create, use and test (1) stereoscopic display technologies, (2) software for 3D content sharing across the internet, (3) motion capture technologies to develop seamless gesture-based user interfaces, and (4) wireless remote controls and touch interfaces based on iPhones, iPod touch devices and iPads. Figure 3.1 provides a snapshot of the different components of the *LINDSAY* project that its development team is currently working on.

3.4 LINDSAY Presenter

A typical scenario for an anatomy lecture for 1^{st} year students works usually like this: The anatomy instructor is standing in front of a class of a few hundred students, elaborating on the human skeleton, muscle, and nervous systems. The instructor is using a plastic model to explain the various muscles of the hand, draws with a pen on rubber gloves to illustrate tissue connectivity, or uses projected images on a screen in front of the class in the form of a slide presentation. This is not an ideal situation for the following reasons:

- Only very few students can actually follow the instructor's explanation, as they sit close enough to see which anatomical structures are being pointed out. Most of the students

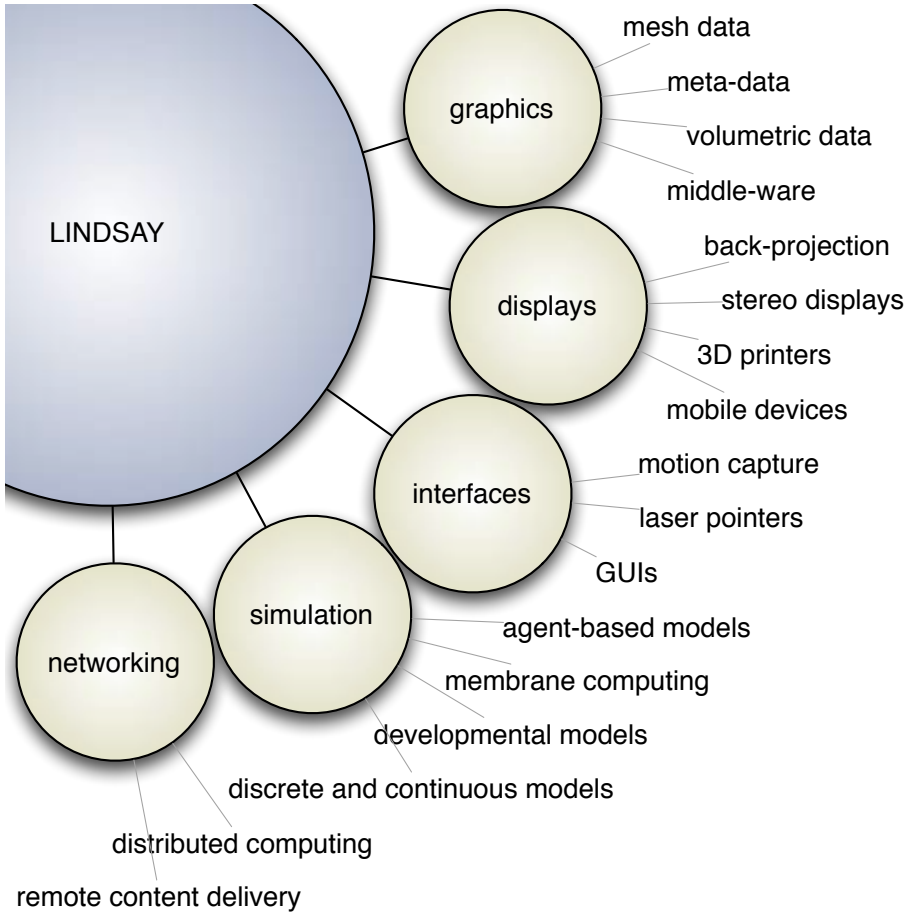


Figure 3.1: As a truly interdisciplinary project, the *LINDSAY* system requires the integration of ideas, techniques and solutions from a wide range of fields within the disciplines of information and communication technologies.

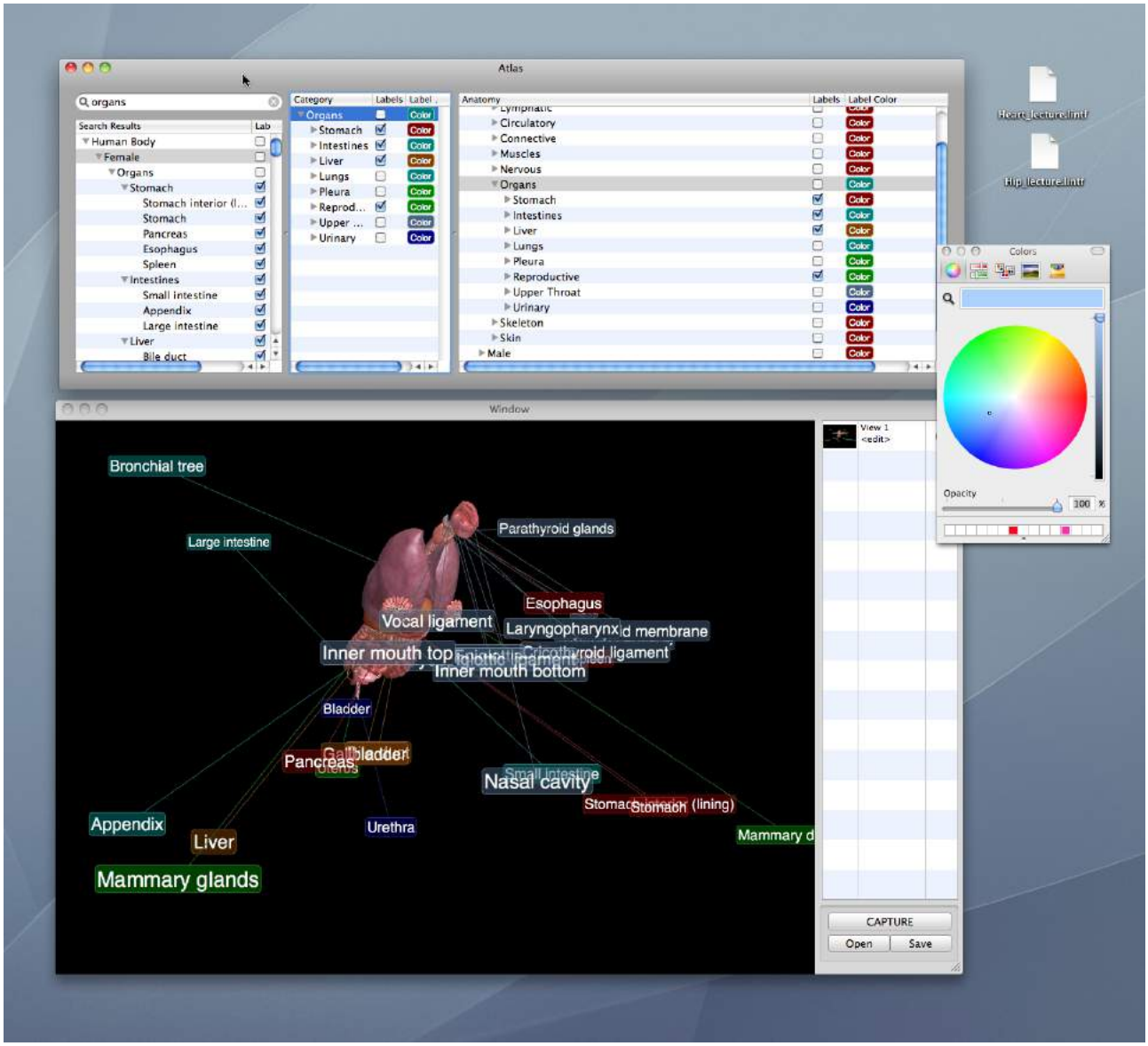


Figure 3.2: The *LINDSAY* Presenter interface with its two main windows: (a) The top window displays the hierarchical list of the male and female anatomy atlas. (b) Below is the main display window, in which the currently selected anatomical structures are visualized.

in the classroom are relegated to what they see on the projected slides, which they might (in many situations they don't!) have also on their laptop screens in front of them.

- The location of anatomical structures within the human body is best understood by getting a true feeling of their arrangements in 3D space. That is, it helps to be able to rotate the body, dissect to reveal hidden structures, and reassemble to get a true sense of each structure's exact position.

- The interrelationships between anatomical structures—such as organs, connective tissues, muscles, cardiovascular or lymphatic networks—is best revealed by, again, studying their points of connection and their distribution across the body.

Obviously, these aspects are hard to illustrate to students without being able to convey the true dimensionality, spatial arrangement and connectivity of anatomical structures. Furthermore, complex structures of the human body and their interrelationships reveal their functions much more easily by seeing them in action and in coordination with other components. Our *LINDSAY Virtual Human* tries to address exactly those issues.

3.4.1 Anatomy Atlas

The backbone of *LINDSAY* is an anatomical database, derived from a male and female anatomy model acquired from Zygote, Media Group, Inc. [295]. The model consists of a set of 3D surface files, organized in folders according to anatomical categories (vascular system, circulatory system, muscles, etc).

However, each file has labels which are often abbreviated and sometimes have no reference to anatomical categories of parts. Using anatomical atlases, we properly labelled and further categorized the parts, storing the data in an XML file. We converted the model to the COLLADA format, which we use as our data interchange format [296]. This facilitates the storage and transfer of not only static model data, but also rigging and information relevant to animation. We import and store the model data in our own binary format, along with the XML file containing the proper labelling and hierarchical organization of anatomical parts.

A snapshot of the user interface to the database is depicted in Figure 3.2. The atlas contains female and male anatomy, which is accessible via a hierarchical list. The selections through

this interface determine which anatomical structures are displayed in the visualization window. Both single and multiple selections are possible, so that, for example, the cardiovascular system (heart and blood vessels) can be displayed in combination with the lymphatic network of the immune system.

The database is also searchable. Search terms can be entered in the field on the top of the left column of the atlas interface. Below this search field, the entries that match the query are displayed. From this result list, further selections can be made to add to the components displayed in the visualization window. The centre column serves as a common-item placeholder. Items from the atlas or search can be dragged in this middle table in order to keep only those components that are needed in a neat and arranged list.

Each entry in the atlas can be annotated with a label in the visualization window. This is illustrated in Figure 3.2, where many of the displayed inner organs are identified by their labels. The labels are arranged in the 3D space around the virtual human. Whenever the model is rotated, the positioning of the labels is adjusted to help with their readablity. As the labels are printed on a transparent background, it is usually easy to read most of them at the same time; if necessary, a small rotation of the anatomical structure would quickly reveal any hidden labels. This greatly improves legibility of the labels and gaining an intuitive sense of which label is associated with which anatomical parts. Organizing the labels and identifying the corresponding parts is also facilitated by different colours, which can be set through the colour chooser window.

3.4.2 Interactivity

The 3D anatomy model can be rotated, moved, as well as scaled—so that structures from the whole body level to the organismal, cellular, and sub-cellular level can be inspected. These changes of perspective can be controlled using keyboard commands in combination with a normal 2D mouse and its control buttons. In addition, we have created a remote control application which runs on iPod touches, iPhones, and on a Pen-Smart pad.

We have also designed prototypical implementations of a gesture-based control interface, where gestures are identified on a video stream from a camera and then translated into navigational commands within the virtual human. Although they were generally well received by

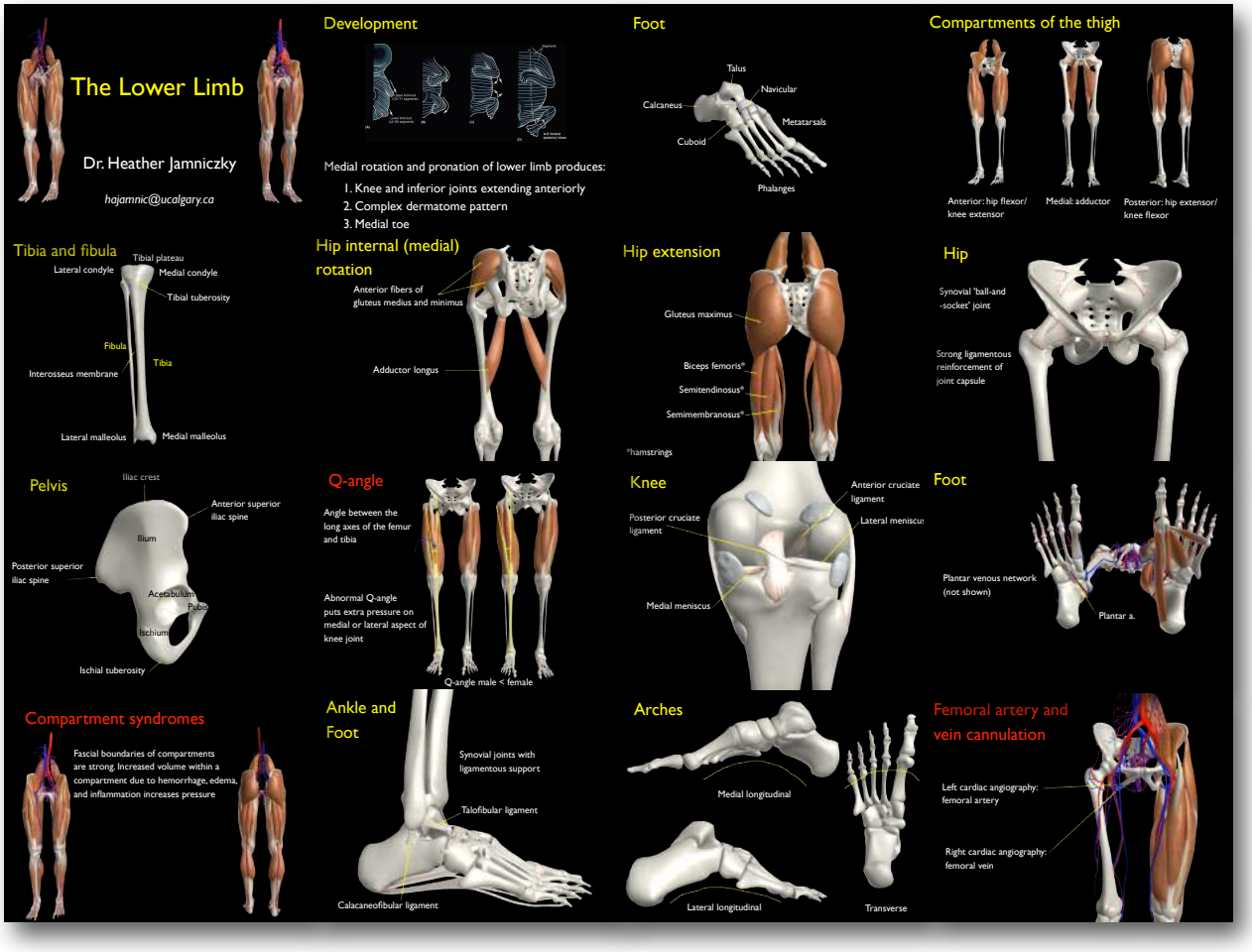


Figure 3.3: Example slide set created with *LINDSAY Presenter*.

the anatomy instructors, they could not (easily) allow for a high degree of interaction, comprising numerous commands. We have conducted experiments with laser pointers that we can use as drawing devices on the projection screen. Here, again, a video camera behind the screen is identifying the position of the laser beam on the screen, translating these into the proper coordinates within the main display window. In the meantime, we have replaced the laser pointer by an iPod touch and iPad, where the drawing is now performed on the touch surface. Figure 3.4(a) shows the navigation screen of our remote control application on an iPhone. Moving a single finger across the touch interface relocates the virtual human in its x-y-plane. Moving two fingers apart or close to each other will increase or decrease the zoom level, respectively. Rotating the iPhone/iPod around its central and lateral axis will make the virtual human turn accordingly. A vertical two-finger swipe moves a customizable cutting surface in and out of the virtual body, which allows one, for example, to see through the ribs, skin and muscles into the inside of the heart. This combination of touch gestures and rotation makes the iPhone/iPod an intuitive remote control device for instructors, who should not need to use the keyboard or mouse for navigation of and around the virtual human.

The iPhone/iPod application can also be switched to atlas mode, where a replica of the hierarchical anatomy ontology is available through the touch interface. Figure 3.4(b) illustrates how this touch interface works. As the iPhone/iPod screen is rather small, it is not practical to display the hierarchical anatomy list all at once. Rather, we opted to show only a single hierarchical level per screen. Selections of anatomical structures are made by double-clicking. Any selections made through the iPhone/iPod interface are immediately replicated on the atlas interface in the main application (Fig. 3.2).

Both through the main *LINDSAY Presenter* and its wireless touch devices, the user can draw simple shapes and lines to enhance or highlight the 3D model currently on display. Custom labels can also be added, which can optionally be connected to specific anatomical structures. These labels are in addition to the already built-in annotations, which display the names of the anatomical structures in the atlas.

(a) (b) (c)

Figure 3.4: The *LINDSAY Presenter* remote control applications running on iPhone and iPod touch devices: (a) The navigation interface allows the virtual body to be moved, scaled, and rotated. Different styles of cuts can 'open' the body. (b) The interface for the atlas gives access to all anatomical structures. (c) Three categories of anatomical parts are selected (indicated by the pink background), which are instantly displayed in the main window.

3.4.3 Creating 3D Slides

As *LINDSAY Presenter* is meant to be used as a presentation tool of human anatomy, we have included an easy mechanism to build sequences of 3D slides. This is similar to the way standard slide presentation software, such as Powerpoint© or Keynote© work; after having created a set of slides, one can progress from slide to slide to enhance an oral presentation. In *LINDSAY Presenter* slides are actually scene descriptions of the 3-dimensional contents; therefore, we refer to these scenes as "3D slides". At any point a snapshot of the current 3D scene can be taken by selecting the `CAPTURE` button at the bottom right corner of the main display window (Fig. 3.5). This then adds a thumbnail in the 3D slide column to the right of the display window. Flipping between 3D slides creates a smooth, animated transition between the associated scenes. A set of such 3D slides can be saved (`Save`) and later loaded (`Open`) into the *LINDSAY Presenter* for subsequent presentation, where the 3D slides can be used as guidance. Figure 3.6 gives examples of such a sequence of 3D slides, here illustrating the anatomy of the female hip. It is worth mentioning, that each 3D slide is, of course, completely interactive, as the "slides" are not simple screen captures of the current 3D contents, but contain the actual scene graph of the current display.

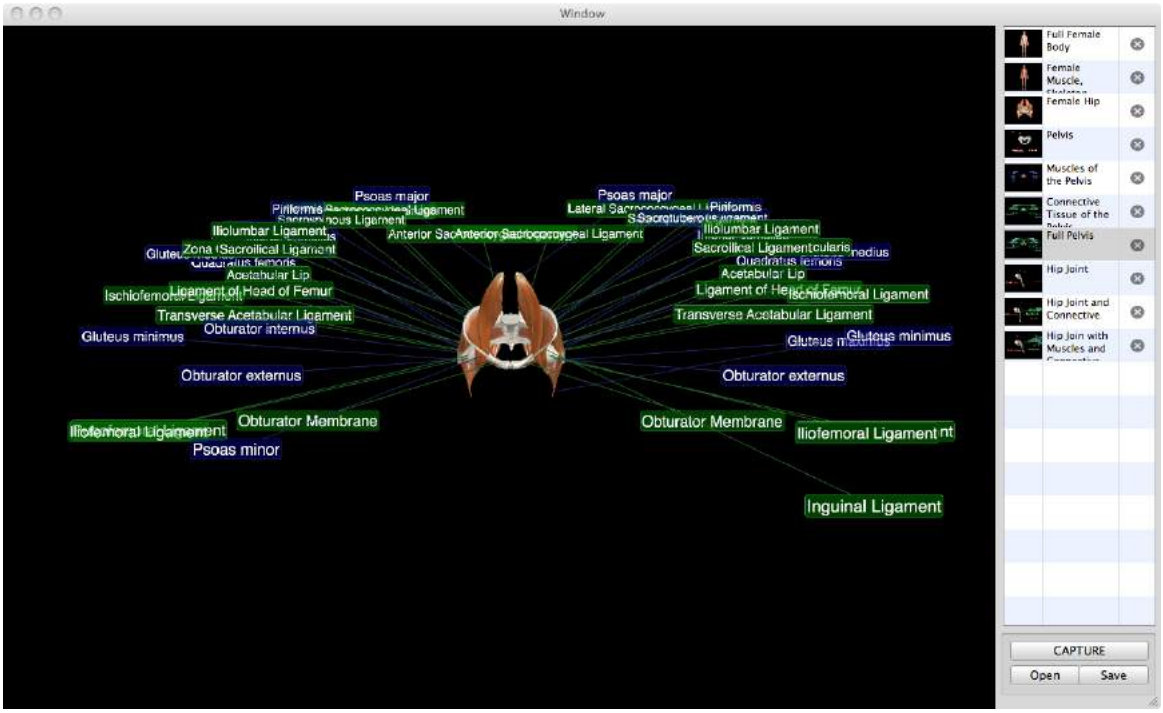


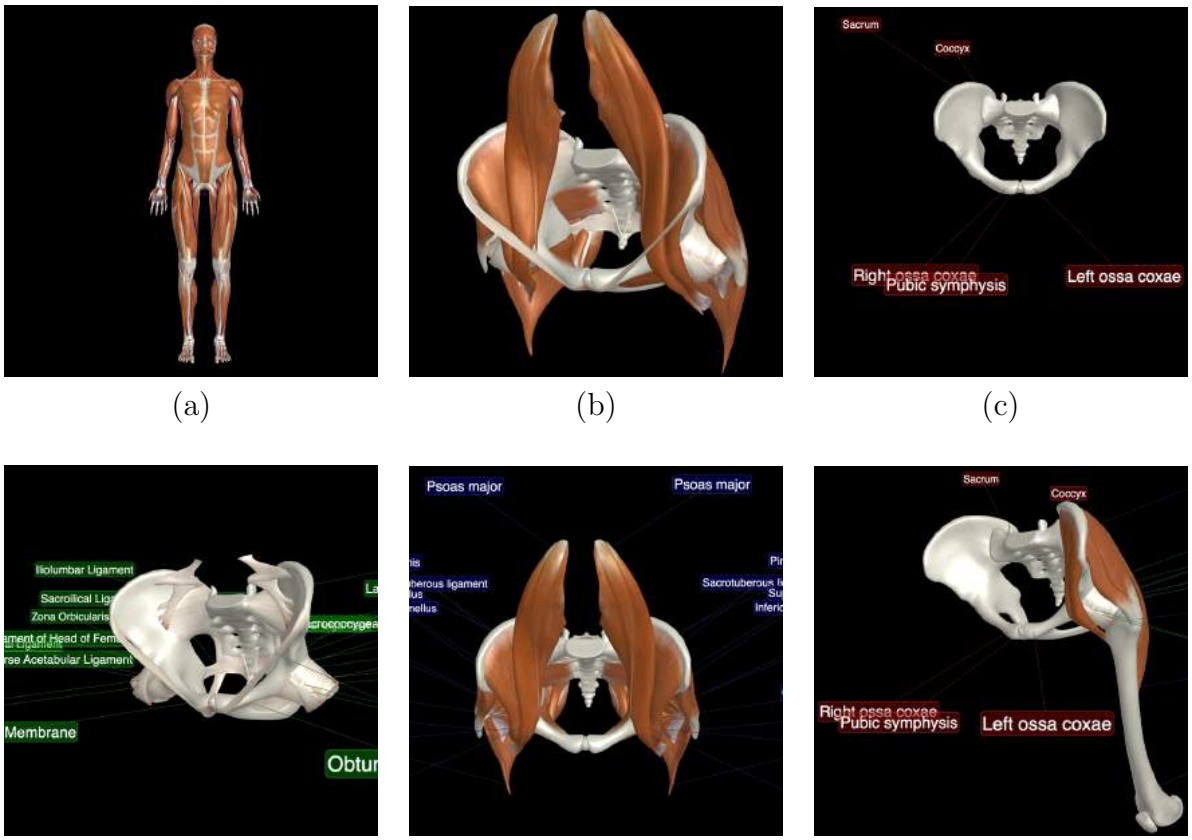
Figure 3.5: 3D Slides in *LINDSAY Presenter*: Demonstration of a typical setup for a lecture, here, as an example, on muscle anatomy. See Figure 3.6 for close-ups of selected 3D slides.

3.4.4 Volumetric Data Integration

We acquired image slice data (Fig. 3.7(a)) of the human body from the Visible Human Project [297]. Using the OpenGL Shading Language, we can render a cut plane of the head from any orientation (Fig. 3.7(b)). Combining the shape of the head with a cut plane, an arbitrary cross-sectional view from any position or orientation can be achieved as demonstrated on the iPhone (Fig. 3.7(c)). We also render using multiple layers to visualize all the slices simultaneously to achieve a volumetric visualization; for this we use the shader again to allow for an arbitrary viewing orientation(3.7(d)). With the application of transparency and colour filters, inner structures can be revealed or isolated (Fig. 3.7(e), (f)).

The Zygote model [295] is stylized and representational, designed to be illustrative of the major structures and well suited for educational purposes. For example, arteries are coloured solid red and veins are coloured solid blue. The Visible Human slice data on the other hand is from photographs taken of actual slices of a cadaver, and often individual organs aren't as readily discernible and surface boundaries not always obvious.

Rendering the data sets together allows for further exploratory abilities. Figures 3.7(d-h) show

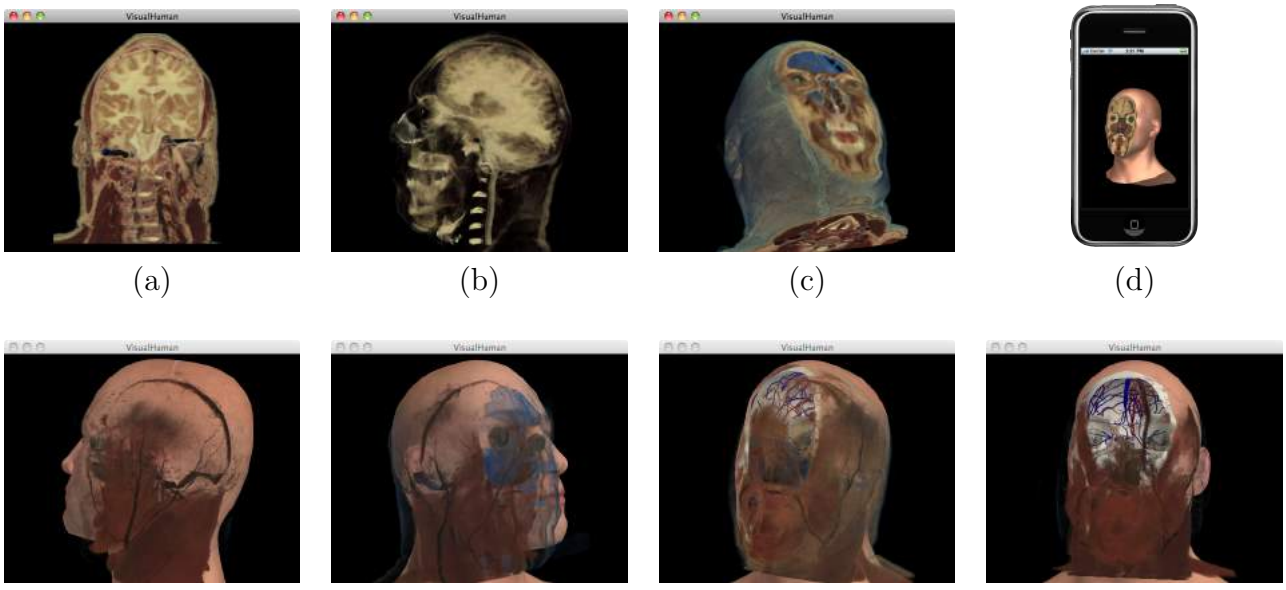


(a) (b) (c)

(d) (e) (f)

Figure 3.6: Examples of 3D Slides in *LINDSAY Presenter*: A typical sequence of 3D scenes that would mark keypoints of navigation through the *LINDSAY* anatomy. Smooth, animated transitions are generated to blend between slides: (a) Female muscle skeleton, (b) female hip, (c) pelvis, (d) connective tissue of pelvis, (e) pelvis muscles, (f) hip joint with muscles.

the slice data superimposed over the 3D model data. Thus, direct comparisons can be made between the representational visualization and the realistic and physically accurate volumetric datasets. In Figures 3.7(g) and 3.7(h) the cut plane reveals the inner structures of the 3D model, while the slice data remains simultaneously overlaid.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.7: Integration of volumetric data from the Visible Human with surface model data. (a-c) Renderings of the volumetric data from the Visible Human data set within *LINDSAY Presenter*; (e-h) the anatomical model data is superimposed on the volumetric data set. (d) The renderings work on both desktop and mobile devices.

3.5 LINDSAY Composer

One of the key objectives for *LINDSAY* is to support the creation of computational models in real-time and combine this with 3-dimensional, highly interactive visualizations of physiological processes related to the human body. *LINDSAY Composer* is a primarily graphics-based programming environment, through which complex biological processes can be compiled using simple drag-and-drop interfaces. Available " building blocks" for such compositions are, for example, fluid fields, cameras, anatomical objects, protein shapes, or cell types. A time line helps to trigger and keep track of events within the simulation, which then allows one to run and replay simulations as needed (compare Fig. 3.12).

Similar to first-person 3D computer games, the *LINDSAY Composer* creates an environment

that immerses the user into a virtual 3-dimensional world, with the context of anatomical structures and processes. Different from prerecorded contents, the underlying computational engines, in combination with the dynamic scene composition, allow for interactive explorations that are guided by the user's intuition and curiosity. Software engineering methodologies, hardware advancements and agent-based models turn simulations into a highly interactive multimedia experience.

3.5.1 The Computational Framework

LINDSAY is being built as a fully functional, 3-dimensional model of male and female anatomy and physiology. Our objective is to use the virtual human to illustrate, for example, which muscles are involved when we run or when we hold a pen. Similarly, we can illustrate and investigate the functional structures and defense processes involved in the human immune system. The immune system, in particular, encompasses complex processes across a wide range of scales: from the organismal level (e.g., the thymus) to the networked lymphatic system (including lymph nodes) down to the cellular (B-, T- cells) and the sub-cellular levels (gene regulation in response to pathogens). Consequently, *LINDSAY*'s modeling framework aims to integrate mathematical and computational models across scales. For example, we use fluid dynamics to simulate blood flow at the whole body level. Using interactive zooming, we can get inside a blood vessel, dive into a capillary or hop onto a blood cell to carry us through the blood stream. At this level we switch to agent-based simulations, where " bio-agents" (i.e., cells, proteins and other molecular structures) are being tracked within the simulated 3D virtual body space. Physics engines, mainly used for computer games, have been adjusted to provide realistic and yet quickly computable scenarios that replicate physiological processes within the human body.

As part of the LINDSAY Virtual Human project, we have developed a component-based computational framework that allows the utilization of various formal representations, computation engines and visualization technologies within a single simulation context. For our agent-based simulations, the graphics, physics and behaviours of our interacting entities are implemented through a set of component engines. We have developed a light-weight client/server component, which spreads its siblings in the system's component hierarchy over a wireless or wired network infrastructure.

3.5.2 Agent-based Modelling

Agent-based models play an increasingly dominant role for the modeling of biological and social systems [298, 299, 300]. We see the most promising potential in agent models that incorporate swarm intelligence techniques [229, 230], as these result in more accurate and realistic models. The use of agents as independent and interacting entities is particularly crucial when spatial aspects play a key role in defining patterns of interaction, in understanding their emergent properties, and in helping to shed light on the inner workings of physiological processes.

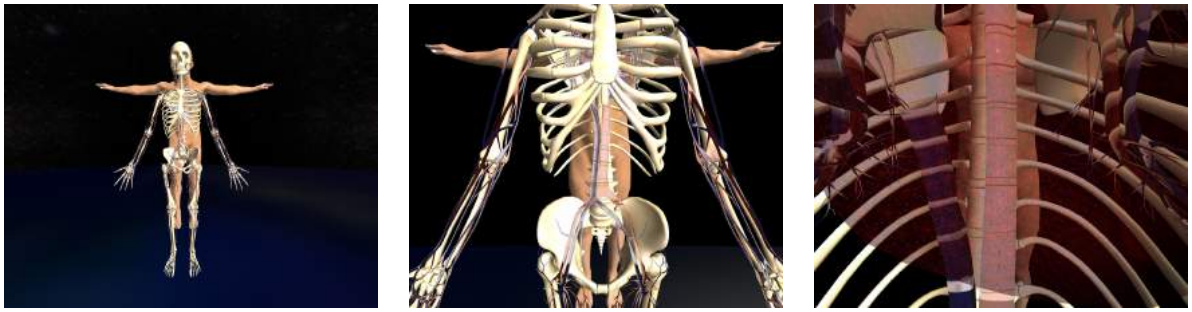
All biological systems, such as the human body, are inherently driven by interaction processes in a 3-dimensional world. Therefore, in order to capture physiological processes within our *LINDSAY Virtual Human*, we utilize swarm-based, 3-D simulations which exhibit a high degree of self-organization, triggered by relatively simple interactions of a large number of ' bio-agents' of different types.

The *LINDSAY* human body is a perfect example that allows for middle-out modeling [301]. Other models, which we have worked on before and are currently incorporating into *LINDSAY*, include the study of chemotaxis within a colony of evolving bacteria [302, 303], the simulation of transcription, translation, and specific gene regulatory processes [304], as well as studies of affinity and cooperation among gene regulatory agents for the λ switch in *E. coli* [289].

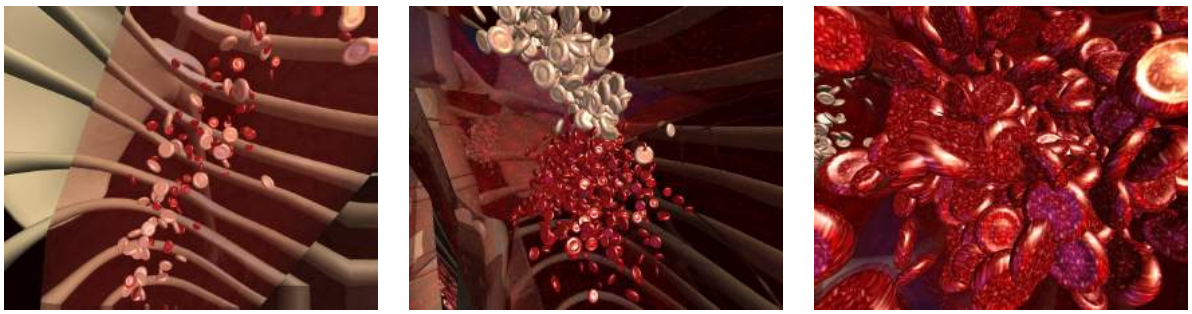
3.5.3 Component Architecture

LINDSAY Composer is written in Objective-C, but is also compatible with C, C++, Python and any other scripting languages that can interface with the Objective-C runtime system. The core framework of *LINDSAY Composer* is as minimal as possible. Having such a simple framework makes it very easy for developers to extend the system with plugins that add new component types, new component engines, and even new user interfaces. Without great efforts, components can collaborate with each other due to the simplicity of a component's API.

In *LINDSAY Composer* a simulation is represented as a hierarchy of components that encapsulate state and behaviour. A component may be registered with a computational engine (Fig. 3.9). For instance, one might have a physics engine managing rigid body interactions, those rigid bodies would be represented in the simulation by physics components. It is not necessary



(a) (b) (c)



(d) (e) (f)

Figure 3.8: Computational modeling of physiological processes across scales by example of the circulatory system: (a) full body view, (b) close-up with upper skeleton, (c) inside the rib cage, (d) approaching the main artery, (e) inside the main artery with red and white lymphocytes, (f) close-up of a cluster of lymphocytes.

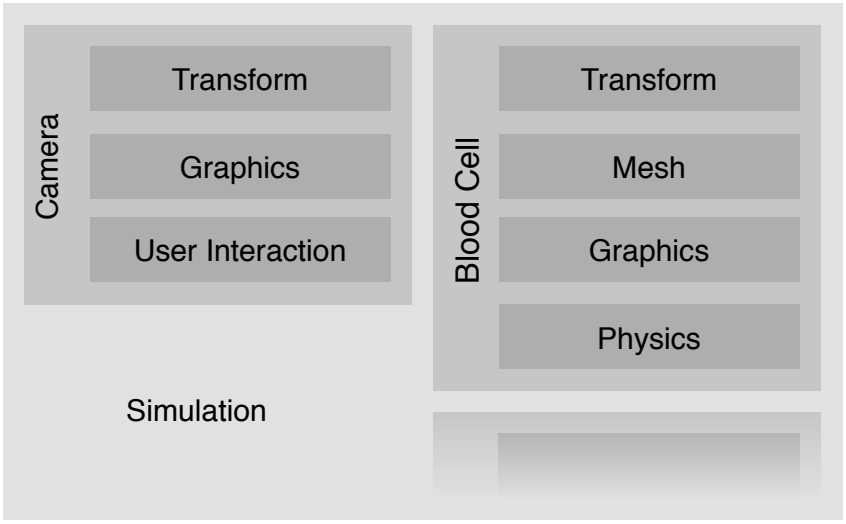


Figure 3.9: A component hierarchy typical of most simulations. Boxes represent components, the embedding of boxes within each other is representative of the component hierarchy, shading is used to exemplify this.

for a component to be registered with a component engine, as is the case with a component that holds only position and orientation data, which we call a transform component.

Components can query their parents for siblings of a given type. Components can also query the entire simulation for components that match a given expression. A common use for this mechanism is for sibling components to share a transform component. For example, one might have a physics component updating the transform, while a rendering component draws a 3D mesh relying on the very same transformation information. In another instance, a camera may be manipulated by a user interaction component that updates a transform component's orientation and position. A graphics component would then adjust the view in the direction indicated.

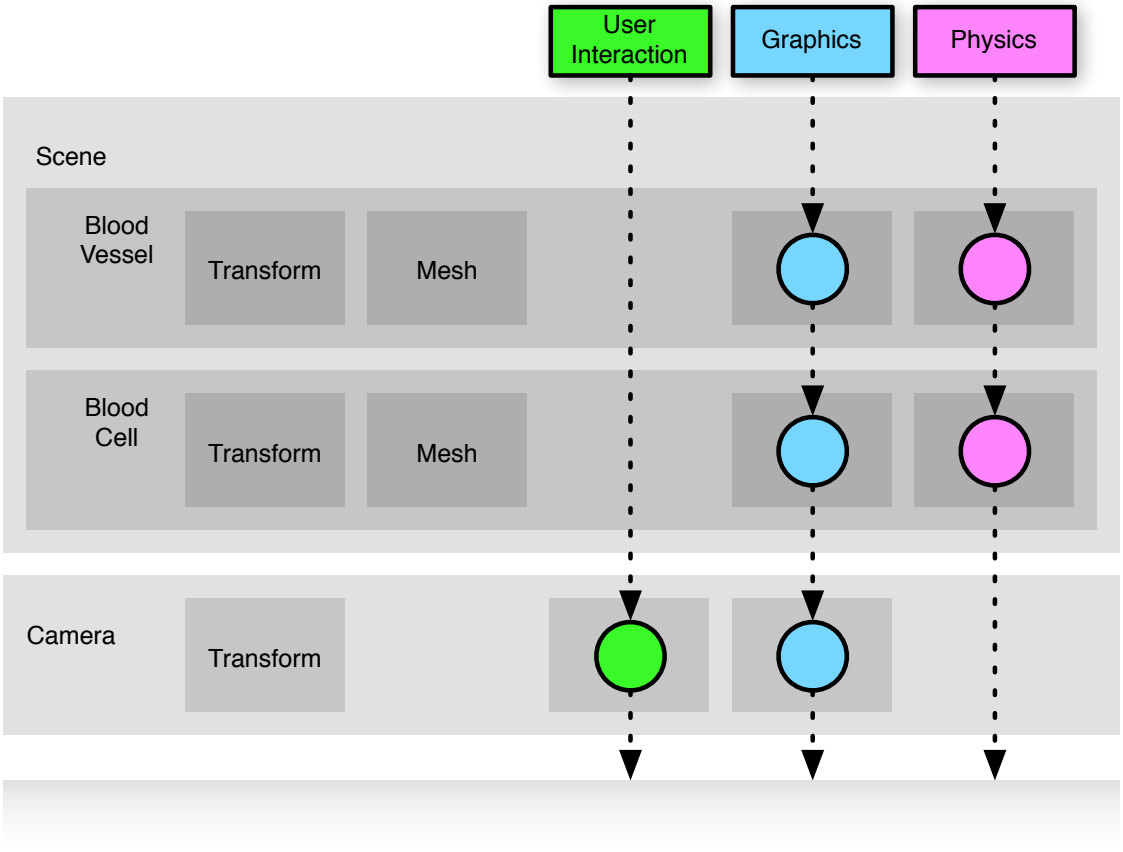


Figure 3.10: Example architecture of component engines: The square boxes at the top are simulation engines for user interaction, graphics, and physics. Elements in the simulation, such as blood vessels and blood cells, contain components for their rendering, interaction, and physical properties. Those components register with the respective engines (denoted by the arrows). Other components, such as transform and mesh, are not registered with a simulation engine. As in Figure 3.9, the nesting of boxes (and circles) corresponds to a components place within the component hierarchy.

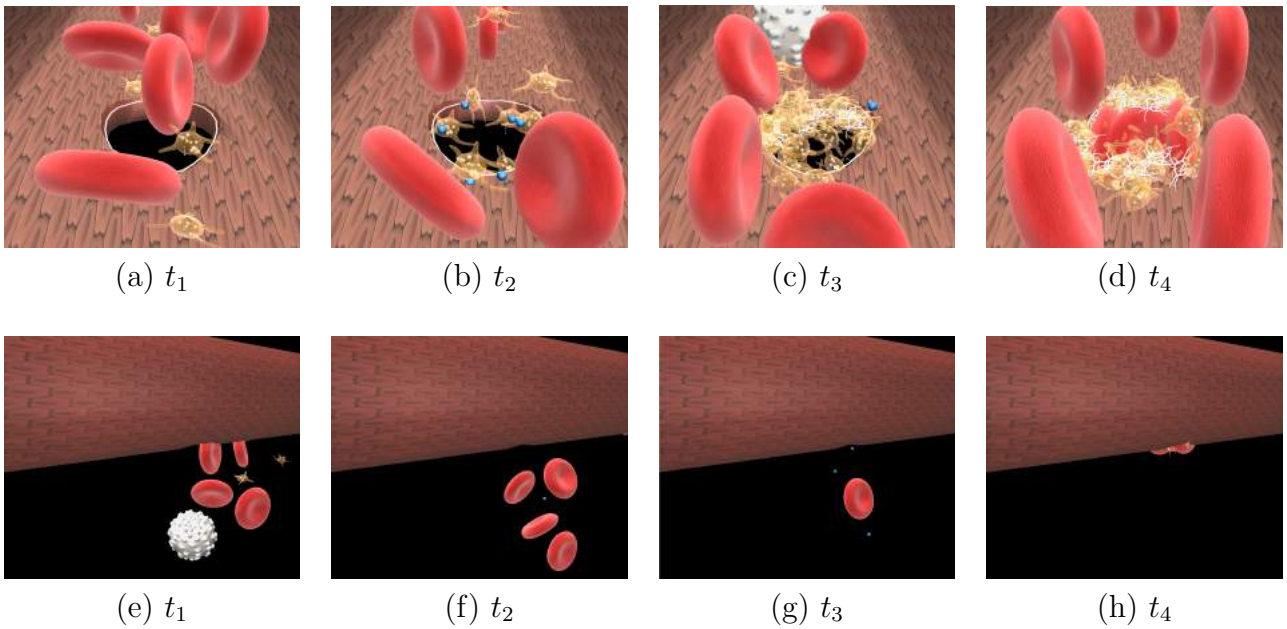
LINDSAY Composer uses a number of engines to form the core computational framework (Fig. 3.10). For instance, a *physics* engine manages rigid body interactions and dynamics, a *graphics* engine renders the simulation to the screen, a *networking* engine handles the distribution of components to other nodes in a network, and an *interaction* engine allows the user to interact with the simulation via mouse and keyboard or multi-touch enabled devices. Each simulation engine iterates through the components it manages to update them. Figure 3.10 shows how components are associated with different component engines while still being part of a component hierarchy.

Especially important for the interactive simulation control as well as for collaborative classroom experiences are the networking components of the *LINDSAY Composer* [305]. A server component is anchored into the simulation's component hierarchy which replicates its siblings and sends them over a network to one or more client components. There can be multiple servers and clients within a single simulation. This allows for distribution of a shared simulation to a number of client devices, which is a typical scenario in a classroom setting. It also allows for interaction with a simulation via mobile devices such as the iPod Touch, iPhone or iPad.

3.5.4 Graphical Programming Interface

In this section we explore the *LINDSAY Composer* user interface. Figure 3.11 shows a blood clotting simulation that we built with *LINDSAY Composer*. The simulation displays two separate views of the same model at different time steps [290]. At the top one can see red and white blood cells, fibrinogens, fibrins, platelets, and other messenger molecules. All of these are implemented as separate agents, that interact with one another and with the blood vessel wall, which is lined by endothelial cells. Over time, one can see that the wound site is eventually filled by platelets and fibrins, which stops the simulated bleeding. The bottom row shows the same simulation from a perspective outside of the blood vessel, which is embedded in the circulatory system of the virtual body (compare Fig. 3.8).

Building a simulation in *LINDSAY Composer* is achieved via a drag and drop interface. Users can drag components from a library of component types into the simulation view, where they will appear immediately if they have a graphical representation. Figure 3.12 shows a typical screen display during the modelling process of a simulation.



(a) t_1 (b) t_2 (c) t_3 (d) t_4

(e) t_1 (f) t_2 (g) t_3 (h) t_4

Figure 3.11: The blood coagulation simulation at different time steps ($t_1 < t_2 < t_3 < t_4$). The process is observed from two different perspectives: inside (a-d) and outside (e-h) of the vessel. Different views are defined by cameras that can be navigated through and are located within the simulations.

In the toolbar, at the top of the window, we see controls for the camera, for bringing up the scene inspector (Fig. 3.13), for the frame rate, and for hiding the template library and timeline views. The user may directly interact with the scene by way of a 3D navigation and selection interface. Navigation and selection is performed via mouse and keyboard, or via networking components through mobile devices with touch interfaces.

In Figure 3.12, the component library is displayed on the left-hand side. From here, components are dragged into the simulation view on the right. Directly below the simulation view is the timeline view. We use a timeline to control the lifetime and properties of components within a simulation. This allows for the dynamic control of a simulation. The horizontal bars in the timeline correlate with the objects that exist in the simulation.

Overlaying the topmost time-bar we see an interpolation graph, which is represented as a component within the simulation. In our blood clotting simulation this graph corresponds to the velocity of the blood within the blood vessel, hence we can simulate the pulsing action of the heart on blood within a blood vessel. These interpolation components can be applied to any compatible property of any component, or set of components within the simulation.

Another important part of the *LINDSAY Composer* interface is the inspection of templates and

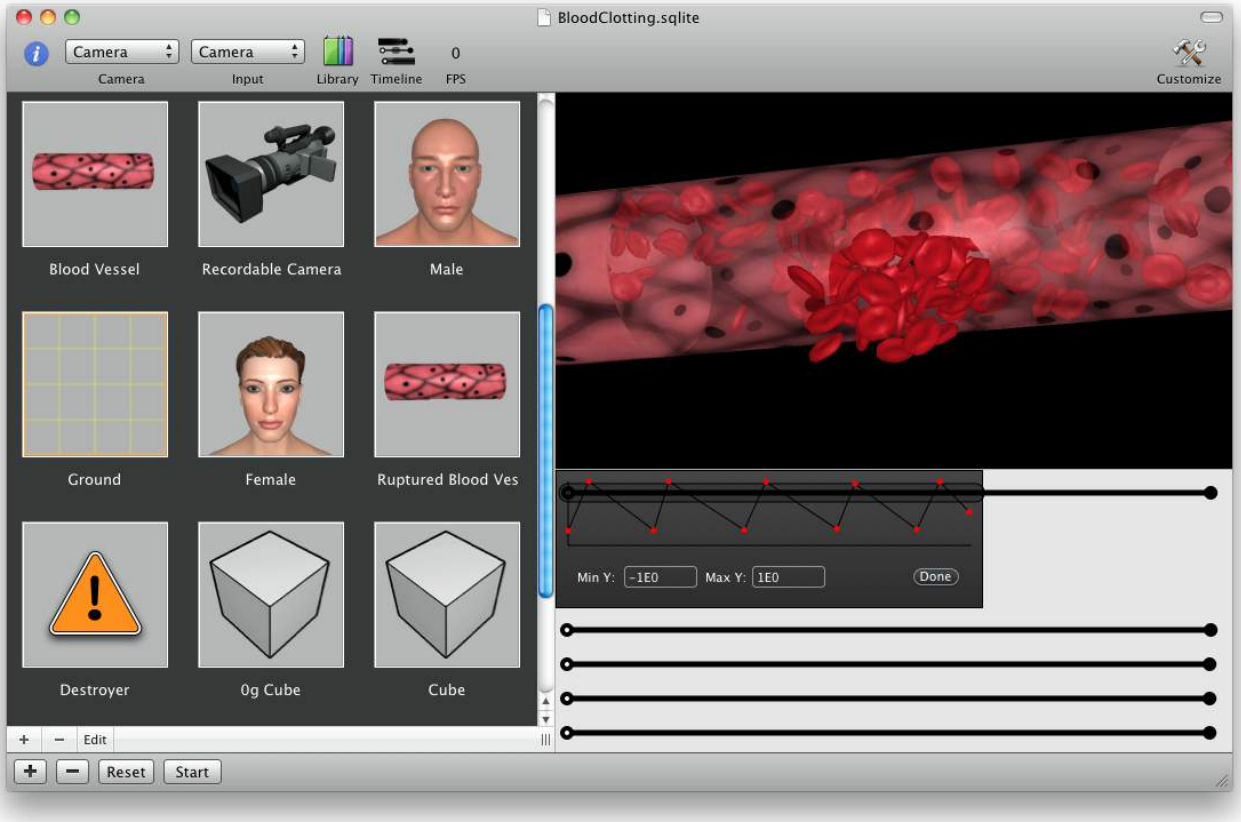


Figure 3.12: Components are dragged from the left into the rendering view on the right to compose a simulation. An overview of the interaction processes is provided in a configurable timeline (bottom-right). The centred overlay window shows an interpolation component that directs the blood flow.

of the hierarchy of a simulation (Fig. 3.13). We dynamically generate the interaction dialogs for a component at runtime. This user interface allows for the realtime inspection and editing of properties as the simulation runs. We also allow for preprogrammed interfaces by checking for their existence at runtime.

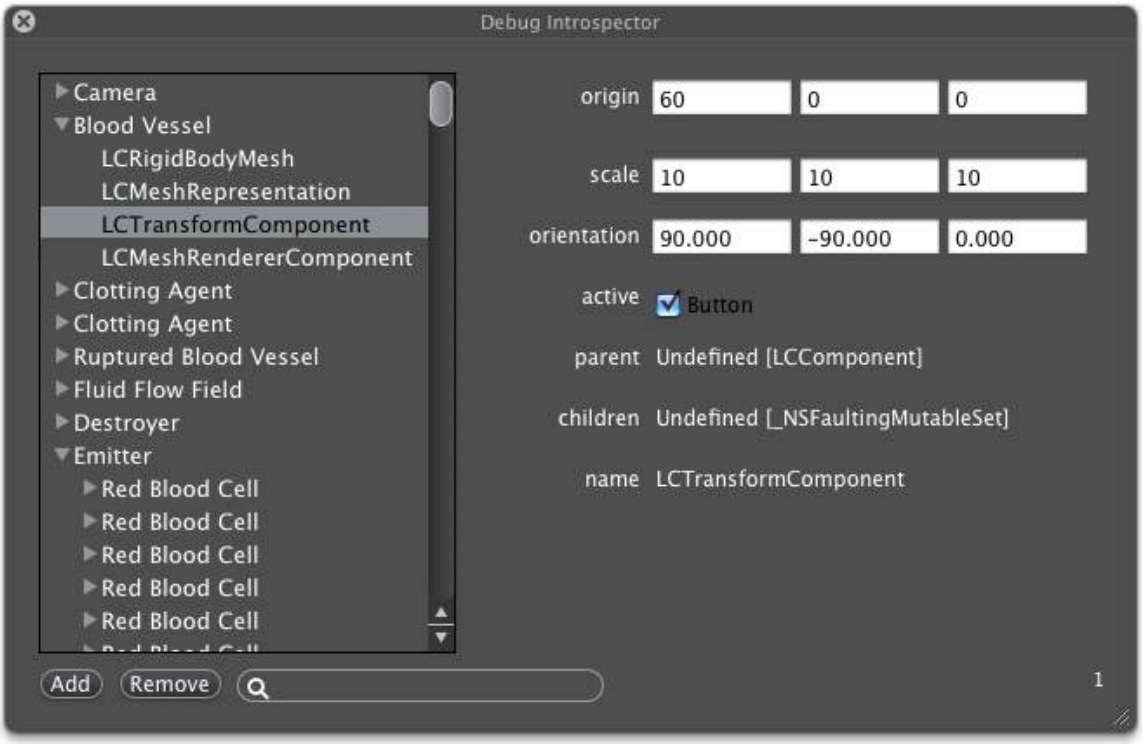


Figure 3.13: (a) The *LINDSAY Composer* GUI for inspecting the scene, its hierarchy, and its components.

3.6 The Educational Perspective

Medicine is replete with complex information spaces. Integrating physiology and pathology, as well as functional anatomy, within true anatomical displays—as opposed to full cartoon images of the body—gives users the opportunity to learn multiple content pieces at the same time. From an educational perspective, we are approaching an application of cognitive flexibility theory and constructivism within a simulation environment that most closely represents true (not cartoon) anatomy.

Putting the tools in the hands of the users allows them to build their knowledge and to construct pathways for their own learning. Each learner is able to pace their learning, and/or create a

physiological or anatomical environment that suits their personal learning needs. Faculty is able to construct, play, pause, rewind, re-create and deconstruct the learning environment in order to emphasize, assess, or modify learning tempo and goals to meet learners' needs.

3.7 Current and Future Work

Interactive Classroom After anatomical and/or physiological scenarios have been created using *LINDSAY Composer*, one can use *LINDSAY Presenter*to run, demonstrate and explore these compositions. We are currently working with instructors from medicine, nursing, kinesiology, and veterinary medicine to turn our software into an effective presentation toolset for the classroom. Using the described client/server architecture, we are now able to create multiple visualization and remote control components that are shared on a wireless network. This lays the foundations to create innovative scenarios for inquiry-based learning and teaching. Using iPods and iPads, we deliver visualizations to wireless devices, which, in turn, can be used to inspect and control different aspects of a shared physiology model, which runs on a powerful simulation server.

Remote Learning and Content Sharing Building an infrastructure for immersive visualization revolutionizes the way human anatomy and physiology is being taught. Such 3D contents can be shared worldwide across the internet by using 3D content sharing technologies, where multiple users can see, "grasp" and interact with 3-dimensional anatomical objects—from organs to cells—and observe the unfolding of simulated physiological processes. This means, contents produced in one location can be distributed to other educational and research institutions across the globe.

3.8 Conclusions

We described the *LINDSAY Virtual Human* project, gave details on its programming frameworks, as well as its display, visualization, and interaction technologies. We also shared our initial experiences about how to bring computer modeling, 3D visualization and human-computer

interaction technologies into medical classrooms. More information on the *LINDSAY Virtual Human* project is available on our website: `http://lindsayvirtualhuman.org`.

Chapter 4

Interactive Multi-Physics Simulation for Endodontic Treatment

In this paper we present a novel approach to simulating dental treatment of root canals. Preceding interactive simulation approaches to dental training focus on preparing access cavities and working on the hard dentin of a tooth. Their common goal is to provide haptic feedback necessary to impart manual dexterity to students. In contrast, we focus on learning about the intricate complexity involved in root canal treatment, considering different root canal morphologies, differences in the texture of the pulp tissue as well as the interaction possibilities offered by different dental instruments at individual steps of the procedure. Due to this shift in focus, new computing challenges appropriate for physical interactions emerge. In this paper we elaborate on recent developments in realtime physics simulation and we demonstrate the backend mechanisms needed to drive more complex dental training simulations, amalgamating different representations for real-time physics calculations.

Sebastian von Mammen, Marco Weber, Hans-Heinrich Opel, Timothy Davison: *Interactive Multi-Physics Simulation for Endodontic Treatment.* In: *Proceedings of Modeling and Simulation in Medicine Symposium at SpringSim 2015*, Curran Associates, Inc., 2015 , pp. 36–41.

4.1 Introduction

Treating the root canal of a tooth becomes necessary if its pulp tissue gets infected. The root canals are cleaned from the pulp, restored with a filling, and the access cavity is closed again. There are more than seven million root canal treatments each year in Germany alone [306]. Yet, it has been ascertained internationally that the rate of success of such endodontic procedures is not satisfactory. In part this might be due to inconsistent protocols [307, 128], partially to accidents during treatment, but mainly it is due to the persistence of infections in the root canal system [308]. The frequent failure to clean root canals does not come as a surprise considering the complexity of root canal morphologies (varying diameters, bifurcations, isthmuses, and dividing and merging canals) [309], the available assortment of instruments, and operation aids (mainly regarding magnification and illumination) [310]. Accordingly, the following quote from [311] stresses the need for system comprehension to achieve successful treatments: "A thorough understanding of the complexity of the root canal system is essential for understanding the principles and problems of shaping and cleaning, for determining the apical limits and dimensions of canal preparations, and for performing successful microsurgical procedures."

Although there have been approaches to virtual simulation and training of root canal treatment, or *endodontic therapy* [312], they were rather limited in their scope, focusing on the first step of the procedure, e.g. [313, 314]. As a first step, the dentist needs to prepare an access cavity to reach the root canals' pulp tissue. This first step is soon learned by dentistry students training on extracted teeth or phantom teeth and it typically does not pose a great challenge. Next, the dentist would clean out the root canals, removing the pulp tissue as meticulously as possible in order to avoid the aforementioned persistence of infections. Loosening the soft pulp material is performed using a dental handpiece equipped with an according bur and by means of endodontic files. In order to thoroughly clean the root canals, they also need to be irrigated.

In this paper, we present an approach tackling these challenging tasks of endodontic therapy in virtual reality simulation from a real-time multi-physics perspective. The concept comprises the model representation, the computation of the interaction dynamics, and especially the coupling between different conceptual physical representations, i.e. rigid and deformable bodies. The remainder of this paper is structured as follows. In the next section, we briefly touch upon

related work including virtual reality approaches developed for dental training and real-time physics approaches. Next, we explain our concept, detailing the process of asset generation, their physical representation and their physical interaction mechanics. Before summarising our contributions and providing an outlook on future work, we discuss our approach in the light of recent advancements in real-time physics simulation.

4.2 Related Work

Work related to our contribution can mainly be aligned with two directions: Virtual simulations for dental training and approaches to real-time physics computation. In this section, we briefly touch upon both these fields, starting with preceding virtual dental training approaches that only rely on rigid body dynamics.

4.2.1 Virtual Dental Training

In the 1990s, DentSim was introduced into the market [129]. It is nowadays used to train students at six universities across the U.S.A. It tracks the students' activities by means of video sensors that pick up signals from LED emitters that are integrated in the students' dental instruments. Based on this data, feedback can be provided about the students' treatment success, while working on phantom teeth. DentSim was the first system of its kind to undergo extensive validation, e.g. [315].

As mentioned in the introduction, a moderate number of computer-based dental training systems was developed that focused on haptic feedback. Among others, a multi-modal setup was presented that established a stereoscopic view by a shutter glass-filtered projection on a mirror just above an operation space [316]. Guiding an instrument for operating on a so-called phantom head was simulated relying on what is nowadays called a Sensable Phantom Omni device, a force-feedback contraption that offers three degrees of freedom at rather small dimensions. PerioSim[317], Voxel-Man [130], or the Virtual Reality Dental Training System [133] all utilise the Omni device to let the student acquire manual dexterity. In the context of endodontics, all these approaches only offer the preparation of the access cavity. Nowadays, there are numerous force feedback devices available and, accordingly, additional haptics-oriented concepts have

been presented [134, 318]. Yet, to the knowledge of the authors there is not a single virtual reality solution that attempts to teach the complexity of endodontic therapy's actual challenge, namely cleaning root canals of complex morphologies.

4.2.2 Real-time Physics Approaches

Interactive simulations offer the means to manipulate model components to various degrees. Depending on the level of abstraction, these manipulations may be offered at the level of graphical objects that can be created, deleted, displaced or otherwise adapted to fit the user's mental picture. In physics-aware model environments, e.g. where several objects may not fill the very same spot, this kind of interaction defines the need for the integration of a physics engine. A comprehensive overview of the taxonomy of the vast field of physics simulation is provided by [319]. However, in the scope of this paper, we focus on real-time methods of forward dynamics, considering three categories: rigid body dynamics [320, 321], deformable body dynamics [322], and particle-based fluid dynamics [323].

At the very foundation, the general laws of motion drive physics engines. Non-penetration constraints, collision resolution and friction forces, and complementary constraints round off the field of rigid body simulation. For calculating the respective forces, various approaches exist, e.g. the penalty force method, Lagrange multipliers, impulse-based simulation, and reduced coordinate formulation. Systematically removing degrees of freedom among model components by introduction of constraints, one can define mechanical joints between rigid bodies. Model constructs that are comprised of a multitude of joined links are referred to as articulated bodies [324].

Deformable bodies can be understood as yet another extension of articulated bodies in the sense that the nodes of a physics representing mesh are all intertwined. An overview of traditional modeling approaches to deformable bodies in an animation context can be found in [325]. Diziol et al. presented an approach to computing incompressible deformable mesh dynamics that is superior to previous real-time approaches in terms of efficiency and accuracy [322]. In addition to various optimisation steps, this approach benefits from the simplifying idea of inferring the effecting forces on individual nodes of the physics mesh from the differences to the original shape of the surface.

Fluids can be computed at interactive speeds relying on the smoothed particle hydrodynamics model [323]. Here, fluid dynamics emerge from the idea that a large quantity of particles interact. The local neighbourhood of each particle determines its individual, virtual density. Differences in density among neighbouring particles result in pressure, which in turn motivates the particles' acceleration and velocity. Recently, a particle-based model was published that promises a unified approach to multi-physics in real-time [172].

4.3 Towards Multi-physics Representation with Rigid & Rigged Bodies

For our first demonstrator of virtual root canal treatment simulation, we make extensive use of the broadly established methods of rigid body physics calculation. We use it in two different ways. (a) To cope with drilling, shaping, filing of hard dentin material. And (b), to trace the deformation of flexible dental files to unearth insights about their interactions in the root canal system. Before detailing both of these aspects, we outline the general interaction scenario of our virtual endodontics simulation.

4.3.1 Interaction Scenario

The asset base of our simulation is comprised of a representation of the tooth-root complex and the dental instruments being used to treat it. The tooth data (Figure 4.1) has been provided from experienced researchers in the field of computer tomographic imagery. It was captured by means of a Micro-CT scanner and achieved a spatial resolution of about 20 indexmicro microm. Mesh surfaces are extracted from the volumetric data using the Marching Cubes algorithm in order to render the tooth in an interactive display. Initially great numbers of vertices (roughly 700.000 in the displayed case) can be reduced to about 100.000 using standard mesh decimation techniques without loosing crucial morphological or structural information.

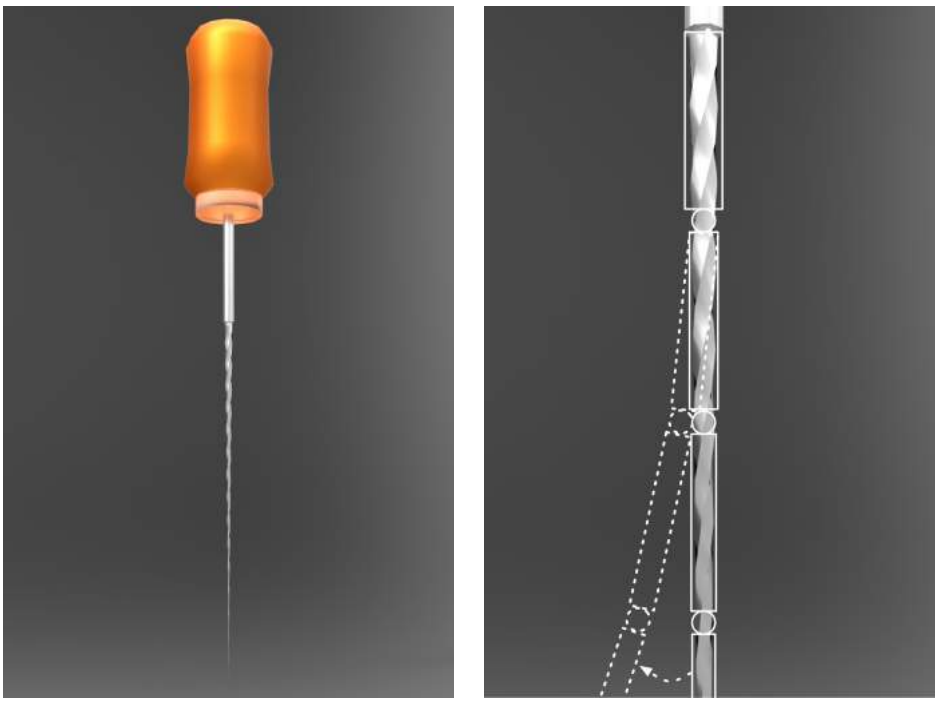
Figure 4.2 shows some of the instruments needed to perform endodontic therapy. We recreated these tools in a 3D modelling application. Towards this end, a simple two-dimensional shape that retraces the cross-section of the tool tip is laid out, segments of its extruded volume are

(a) (b)

Figure 4.1: ISO Surfaces extracted from volumetric CT data of a molar. (a) An opaque material emphasises the coarseness of the surface structure. (b) A transparent material reveals the morphology of the internal root canals.

aligned on top of each other and deformed using twist, skew and proportional scaling modifiers. Figure 4.3(a) displays an according three-dimensional model.

Figure 4.2: From top to bottom: Two files that are manually operated to remove infected pulp tissue as well as one bur that works with a motor-driven handpiece.



(a) (b)

Figure 4.3: The 3D model of an endodontic file. (a) The handle and the geometry built from the cross-section of the instrument's tip. (b) A conceptual display of the instrument's physical representation as a rigged body—stress is propagated along an array of box colliders linked by socket joints.

4.3.2 Drilling, Shaping & Filing

As shown in preceding works on virtual dental training, the volumetric tooth data informs the physical interactions with dental instruments [326]. In order to speed up the physics calculations, we focus on improving the detection of collisions between instruments and the tooth matter. Distance fields provide the core technology for our implementation [327]. Here, the distance field information tells us how far away from a dentin or pulp tissue surface the dental instrument is, or alternatively, how far it has penetrated the according substance. The distance field also provides an efficient data structure to calculate the contact normal as well as the speed of the incident. A penetration event can only occur, if an according force was applied [328]. In order for the user to adjust the applied force appropriately, he might work with one of the aforementioned force-feedback devices or react to corresponding visual cues [329]. In case a sufficiently great force is applied, we remove the intersecting area from the uniform voxel grid and we update the local distance values. The distance field is generated following the concept of level-set segmentation which ensures that the irregularities of the geometry of the tooth, i.e.

bumps and concavities, are considered but scalable patterns are efficiently and parametrically represented [330]. Next, the level-set segmentation is translated into the distance field for the tooth and efficiently used for testing and resolving collisions [331]. There is also a GPU-based implementation for this approach [332].

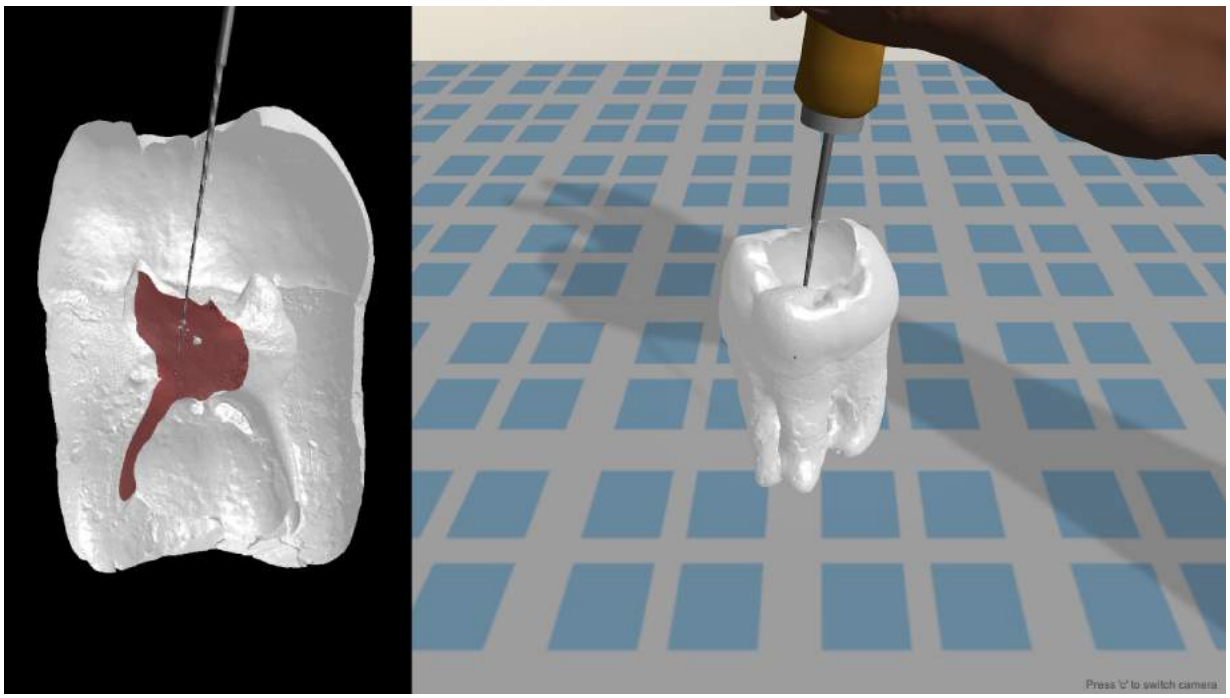
4.3.3 Tracing File Deformation

For preparing access cavities, as has been the usual goal for computer-based endodontics simulation so far, manipulating the dentin voxel volume is the target of any physical interaction. However, especially in case of cleaning the root canals, it is essential to also consider the impact of the physical interactions on the dental instruments. For our first demonstrator, we approached this challenge by fitting an array of box colliders around the endodontic files with a joint connecting each collider to its neighbour (Figure 4.3(b)). The spring dampening coefficient of the joints define the overall body's reactive stiffness, whereas a steady force is working on each link to recover the original shape. This recovery force considers the file material, the angular deviation at the joints and it points towards the central axis of the file. Screenshots that visualise this interaction from within our simulation are presented in Figure 4.4.

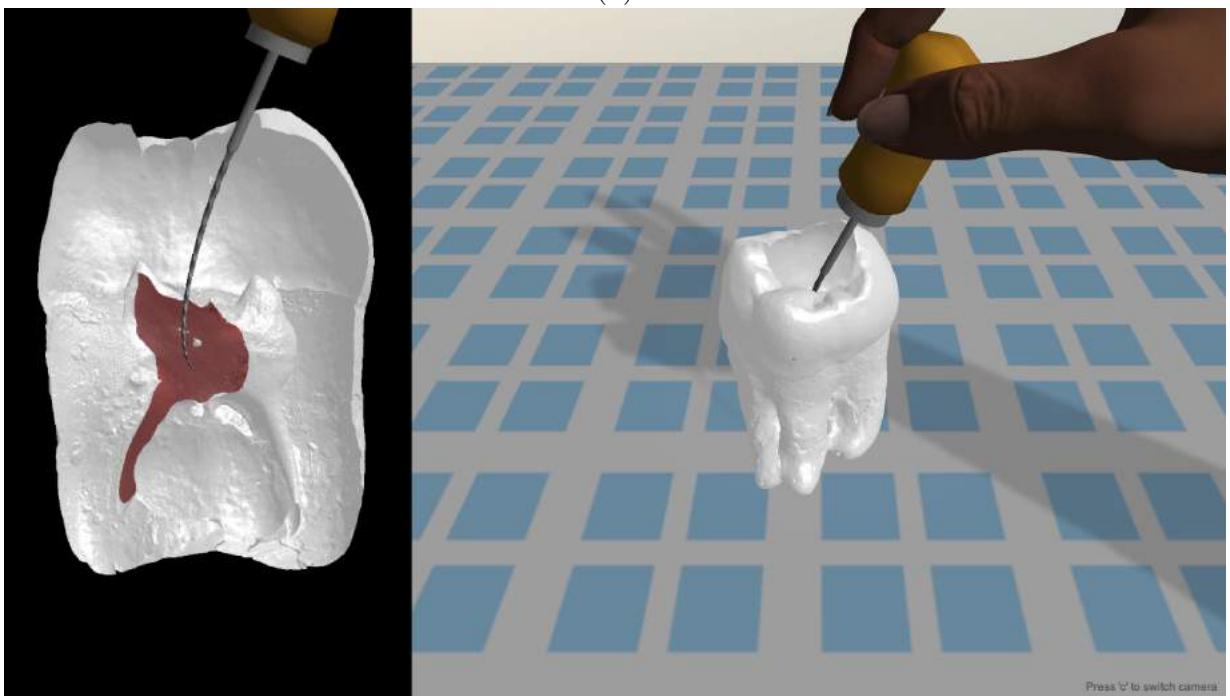
4.4 Discussion

The effectiveness of our current demonstrator has been confirmed by the domain expert on our team as well as by closely collaborating university researchers in endodontics. Its hardware setup features an Oculus DK2 head-mounted display and a LeapMotion finger tracking sensor. In order to increase its accessibility, we need to make it less dependant on special hardware and desktop PCs. Instead, we are in the process of developing a head-mounted VR solution based on ZEISS VR One or Samsung Gear VR, which feature top-of-the-line smart phones and tablets for computing and visualising.

In this light, due to the comparably low processing power of mobile devices, the efficient calculations of the presented approach based on distance field-based collision detection and joint-based deformation are all the more important. However, our concept suffers from certain drawbacks. The greatest ones are the lack of fluid simulation and the strong limitation of the deformable



(a)



(b)

Figure 4.4: Screenshots from within our simulation environment. (a) The endodontic file penetrates soft pulp tissue. (b) The file grinds against the root canal and bends as a result.

dynamics implementation. The latter works great for simulating the flexibility of the dental tools. However, it does not suffice to simulate deformations of pulp tissue. As the scraped off tissue sometimes needs to be flushed out, fluid simulations are yet another important aspect that we have not considered, yet.

Our aim for the next demonstrator is to translate our models into a particle-based representation similar to [172]. For this to happen, we will segment the volume data of the tooth into its different structures, and then manually classify their material properties. We will then use a surface reconstruction algorithm (Marching Cubes [333] or Dual-Contouring [334]) to generate the surface particles. To generate particles beneath the surface, within the volume of a tooth structure, we simply generate particles for each sample-point on a uniform grid. For soft-bodies we can use the density information from the volume data to parameterise the softness of the particles.

4.5 Conclusion & Future Work

Based on seminal dental research literature, we motivated the need for understanding root canal morphology. We presented an approach to address this challenge by augmenting the information typically available in a dental clinic in a virtual training environment. Based on expert feedback, we determined that such augmentation has to happen in the context of the procedural steps that would be performed during clinical therapy. In contrast to preceding virtual dental training simulators, we understand that only a mature multi-physics approach can allow for the expected rich set of interaction dynamics. We have taken the first steps towards this goal, integrating rigid and deformable body dynamics for the targeted application domain. In order to allow for richer calculations, we introduced a level-set-based method to the domain for fast collision detection using signed distance fields. In order to address the need for deformable body physics, which is essential in endodontic simulation, we introduced an efficient, if specialised, method to simulating flexible dental files. In particular, we provided a rigged body structure aligning several box colliders along the geometry of the file. The discussion revealed that our approach still has certain drawbacks. Yet, at the same time, we have already outlined a plausible path to generalise the presented deformable body concept and to introduce and couple the existing physics representations to fluid mechanics.

Chapter 5

Swarm-based Computational Development

Swarms are a metaphor for complex dynamic systems. In swarms, large numbers of individuals locally interact and form non-linear, dynamic interaction networks. Ants, wasps and termites, for instance, are natural swarms whose individual and group behaviours have been evolving over millions of years. In their intricate nest constructions, the emergent effectiveness of their behaviours becomes apparent. Swarm-based computational simulations capture the corresponding ideas of agent-based, decentralized, self-organizing models. In this work, we present ideas around swarm-based developmental systems, including swarm grammars, a swarm-based generative representation and our efforts towards the unification of this methodology and towards improving its accessibility.

Sebastian von Mammen, David Phillips, Timothy Davison, Heather Jamniczky, Benedikt Hallgrímsson, Christian Jacob: *Swarm-based Computational Development*. In: *Morphogenetic Engineering: Toward Programmable Complex Systems*, *Series: Understanding Complex Systems*, ch. 18, pp. 473–500, Springer Verlag, 2012.

5.1 Introduction

Arithmetic operations drive computational processes by updating existing variables or by infering new ones. The selection of operands determines a topology of dependencies among data

which may change over the course of a computational process. In particular, intermediate computational results may control the flow of directives, their sources and their targets. Ultimately, the purpose of a computational process is to create or change information in accordance with goals such as data storage and retrieval, communication, content creation, or data validation, visualization and prediction (simulation).

Computational *representations* are abstraction layers which connect particular models from a scientific domain (domain models) with corresponding models for a provided simulation platform (platform models) [335]. Similar to *knowledge representations* [336], special representations have been studied and designed in the context of simulating developmental processes such as the formation of molecular structures [337], growth and proliferation of cell populations, and structural developments at the organismal level [338].

Not only do these respective computational developmental representations serve different modelling domains but they also rely on various mechanisms of abstraction. L-systems [338], for instance, emphasize the formation of structure based on the generation of symbolic sequences by means of grammatical substitution [339]. Cellular automata (CAs), on the other hand, which are also considered one of the first developmental representations, focus on pattern generation through state changes [340]. Although mitosis and cell differentiation provide a scientifically adequate discretization that bridges from the domain model to the platform model, CAs and L-systems primarily target development at the cellular level.

Focus areas of developmental models have been simulations of the life cycle of cells as well as molecular and intercellular communication—touched upon by CAs and further explored as random boolean networks (RBNs) [341]. Spatial reconfiguration of cell colonies, e.g. through polarization and migration, and thereby changes in the interaction topologies, also play significant roles in developmental systems [342].

In this chapter, we present our work on swarm grammars (SGs), a developmental representation that we have introduced to explicitly combine the ideas of established developmental representations with those of artificial swarm systems. In particular, production rules drive the life cycle of agents (representative of molecules and cells), while the agents' reactivity and motility continuously change the interaction topology of the system. In the next section, we briefly outline work that relates to SGs. Section 5.3 presents various swarm grammar representations that we have designed over the years as well as means to breed SG configurations through

evolutionary computation [343, 344, 345, 346]. In this context, we also present SG examples in the domains of art, architecture and biology [347, 348, 349]. Section 5.4 summarizes and concludes our work.

5.2 Related Work

In the following, we want to build a terminological hierarchy based on *patterns*, *structures*, and *morphologies*. A pattern is commonly defined as "an arrangement or sequence regularly found in comparable objects or events". A structure is "the arrangement of and relations between the parts or elements of something complex". A morphology "[...] deals with the form of living organisms, and with relationships between their structures" [350]. Morphologies describe the structures of organisms, whereas the recognition of their structural patterns reveals insight into the complex arrangements of the parts of an organism. Consequently, the borders blur between patterns and complex structures when facing the challenges of morphological engineering. Reductionist [351] and quantitative [352] analyses of morphological processes necessitate identification, measurement [353] and simulation of complex, emergent processes [335] and integrating, multi-scale models [301]. Exploring these scientific paths is exciting and important. In this chapter, however, we present our findings that are mainly concerned with swarm grammars as a unified swarm-based developmental representation.

5.2.1 Complex Patterns through State Changes

In cellular automata (CAs) lattice grids are populated with cells that change their states—frequently represented as a binary digit—in accordance with their neighbours [340]. As underlined by Giavitto et al.'s categorizations [354], CAs are dynamic with respect to their states, but not in regard to their interaction topologies. Thus, the development in CAs is limited to state-based pattern formations. These patterns, however, may be seen as structure formations nevertheless. Wolfram introduced four classes of complexity for patterns generated by the state evolution of one-dimensional CAs [355]: Those that converge (1) to a homogeneous state (corresponding to limit points), (2) to a heterogeneous state (corresponding to limit cycles), those that exhibit (3) chaotic behaviour (corresponding to chaotic attractors), and those that are (4)

self-organizing, reaching attractors of arbitrary complexity from random initial conditions.

5.2.2 Complexity Measures

Several aspects pose starting points to reveal the complexity of a (computational) model, or the lack thereof. Shedding light on the formation and evolution of high-order life forms, Schuster summarizes various approaches to measure complexity [261]: (1) Ecological diversity can lead to systems occupying niches and yield involved food webs. (2) Construction processes can add to the complexity of a system by providing additional functionality such as providing shelter. (3) Formally, logical depth can also measure a system's complexity. Schuster relates systemic hierarchies (from genes over cells to organisms) to logical depth and emphasizes its relevance for biological systems. Hornby aims at the very same idea, introducing the scalable metrics of modularity, reuse and hierarchy (MR&H), which he applies it to measure structure and organization [353]. In the given context, scalability implies that the complexity of a system increases with size. In a series of experiments, Hornby was able to show that multiplication of the MR&H metrics and normalization by either the design size or by the algorithmic information content (AIC), which accounts for the shortest program to produce a given outcome, yield the desired scaling effect in complexity.[1]

5.2.3 From Life-Cycles to Structure

Although we have introduced the notion of categories of complexity in the context of CAs (Section 5.2.1), L-systems, too, can be subjected to structural measures as described in Section 5.2.2). Lindenmayer and Prusinkiewicz designed L-systems in order to retrace the growth of bacterial colonies and plants [338]. L-systems are a formal system that combines the productivity of formal grammars with geometrical information to direct and translate simulated development into three dimensions. In particular, symbols that encode a geometrical command such as L (left), R (right) or F (forward) are substituted in parallel in accordance with a set of production rules. This is supposed to retrace the developmental stages of a naturally growing structure. Special characters such as $\lfloor$ or $\rfloor$ that are part of the substitution strings introduce

[1] R_a, the average reuse of symbols during program execution works well as a structural measure when normalized against the design size, whereas R_m, the average reuse of modules, yields a scalable measure when divided by the system's algorithmic information content [353].

compartmentalization, i.e. hierarchical information, into the structural outcome. Generally speaking, L-systems loose the appeal of universality that can be claimed for CAs by introducing specialized operators that conduct structural development in an iterative fashion.

At the same time, these geometrical directives render L-systems convenient for describing structural development of natural systems such as plants [356, 357], including simulated plants that interact with their environment [358, 359, 360]. Original work in genetic programming of L-systems [361, 362, 363] has led to several platforms for L-system evolution [364, 365, 366] and the breeding of virtual plants in a coevolutionary scenario, which even displays competitive arms-race situations [367]. Beyond plants, L-systems have also been used to evolve virtual creatures and their control networks [368, 369] as well as for the reconstruction of retina and blood vessel structures [370, 371].

The appeal of CAs is that the interaction topologies remain fixed, while patterns develop based on state changes over time. In L-systems the substitution of existing symbols effectively results in cell differentiation (state changes), the creation of new or the deletion of existing symbols. Thus, the neighbourhood topology can be altered as well as the next production step in the case of context-sensitive L-systems. However, changes in the interaction topology in L-systems are limited to the symbols' immediate neighbours. When modelling molecular diffusion or cell locomotion and migration in an agent-based manner, interaction topologies undergo great dynamics (e.g. [342]). Giovatti et al. termed such systems D^2S in which both the states as well as the topologies are dynamic [354].

5.2.4 Breeding Solutions

The great degree of freedom with a D^2S system brings about the challenge of a largely extended configuration space. Evolutionary algorithms (EAs) are a means to find sets of diverse, good solutions in such large search spaces. We applied genetic programming techniques (GP) to breed swarm grammar systems interactively [343] through the *EVOLVICA* genetic programming framework [365]. We bred swarm grammars like a gardener in a three-dimensional, immersive environment [344]—watering, weeding and recombining individual specimens that grow in a shared environment. Most recently, we let swarm grammars evolve that generated diverse and interesting architectural models [345]. We describe these three evolutionary ap-

proaches in detail as part of the following section which presents different extensions of swarm grammars.

5.3 Swarm Grammars

Our first swarm grammar systems were composed of two parts: (1) a set of *rewrite rules*, which determine the composition of agent types over time, and (2) a set of *agent specifications*, which define agent-type specific parameters that govern the agents' interactions [343]. Next, we assigned the required genotypical information and the rewrite rules to individual agents which allowed for co-existing and co-evolving swarm grammars [344]. At that point, we identified the distinction between agent behaviours and rewrite rules as an artificially created artefact which we had to overcome [345]. As a result, the individuals' rewrite rules were extended and turned into general agent rules [54, 55], including the special ability to create new agents or construction elements and to remove existing ones. In analogy to biochemical processes of secretion and diffusion [301], we refer to these abilities as *metabolic* operations. Lastly, in order to make swarm-based modelling accessible to non-computer scientists, we have been pushing toward a standardized swarm-based modelling and simulation framework [346]. In the latter representation, the relationships among swarm individuals are emphasized and the swarm agents' behavioural rules are streamlined and expressed in graphical notation. In this section, we are going to present these different stages of swarm grammars and illustrate their respective features. A brief overview of these stages is shown in Table 5.1.

representation	*motivation*	*example*	*section*
basic swarm grammar	swarm dynamics + growth	agent-agent and agent-environment interactions, artistic exploration through interactive evolution	5.3.1
decentralized SG	individual behaviours	artistic exploration through interactive evolution	5.3.2
decentralized, rule-based SG	event-based interactions	breeding architecture through automatic evolution	5.3.3
swarm graph grammar	improve accessibility and standardize simulation	simulation of biological developmental processes	5.3.4

Table 5.1: Four evolutionary stages of swarm grammars.

5.3.1 Swarm Grammars with Centralized Population Control

A basic swarm grammar system $SG = (SL, \Delta)$ consists of a rewrite system $SL = (\alpha, P)$ and a set of agent specifications $\Delta = \{\Delta_{a_1}, \Delta_{a_2}, ..., \Delta_{a_n}\}$ for n types of agents a_i. The rewrite system SL is a probabilistic L-system with axiom α and production rules P, as described in [338] and [365]. In the simplest form of context-free $0L$-systems, each rule has the form $p \xrightarrow{\theta} s$, where $p \in \Omega$ is a single symbol over an alphabet Ω, and $s \in \Omega^*$ is either the empty symbol (λ) or a word over Ω. The replacement rule is applied with probability θ. Each agent a_i is characterized by a set of attributes, Δ_{a_i}, which can include its geometrical shape, colour, mass, vision range, radius of perception and other parameters that determine its overall dynamics and interaction behaviour.

The Swarm Agents' Interactions

Figure 5.1 depicts a common view of a swarm agent in three-dimensional space. We configure the local flight behaviour of an agent according to a simple ' boids' model as it comprises the general ideas of local and global attracting and repelling forces [1]. More specifically, at each simulation step, an agent's acceleration vector $\vec{a}$ is set to a weighted sum $\sum_{i=0}^{4} c_i \vec{v_i}$, with $c_i \in [0; 1]$ being the weights for normalized vectors $\vec{v_0}$ to $\vec{v_4}$ that result from the computations of separation, cohesion and alignment urges among local agents as well as from the individuals' drive towards a global target, and the consideration of some noise. The weights c_i are part of the individuals' genotypes as they determine their flight behaviours.

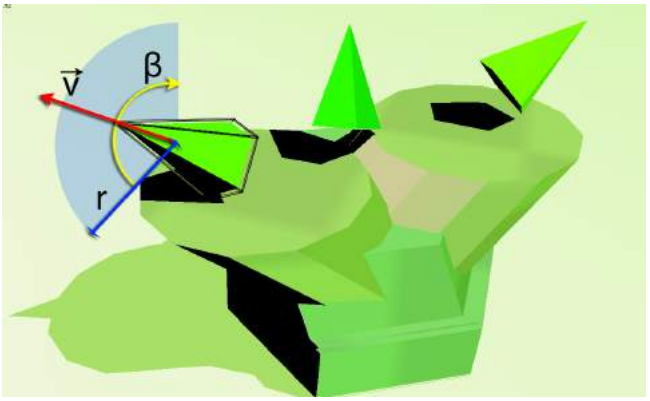
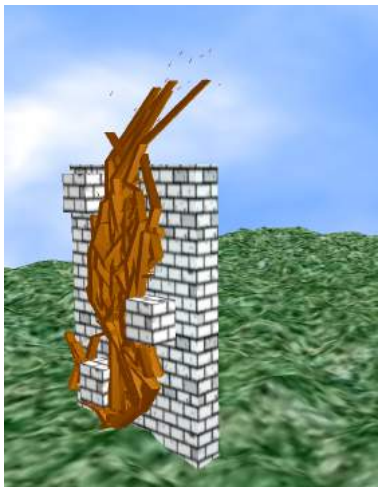
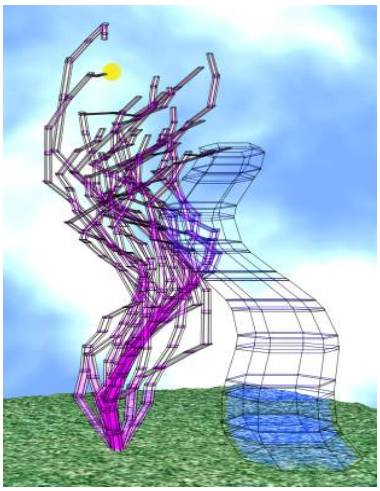


Figure 5.1: The swarm agents are typically represented as pyramidal cones oriented towards their velocity. An agent's field of perception is determined by a radius r and an angle β.

$SL_a = \{A, \{A \rightarrow BBB, B \rightarrow A\}\}$
(a)



$SL_b = \{N, \{N \rightarrow NN\}\}$
(b)



$SL_c = \{I, \{I \rightarrow I\}\}$
(c)

Figure 5.2: Swarm grammar agents interacting with their environment and their corresponding swarm rewrite systems.

As soon as it has run out of energy, an agent stops acting and is not considered by the SL-system rules any longer. Energy levels are inherited through replication. The energy level also influences certain properties of the built 3D structures such as, for example, their size.

Several values characterize the construction elements or building blocks that are placed in space by the swarm agents after it has flown for a certain number of iterations I_d. The shorter these intervals are, the smoother the appearance of the emerging construction. The colour and the numbers of edges define the design of the cylindrical shapes.

For example, a swarm grammar $SG_a = (SL_a, \Delta_a)$ with

$$SL_a = (\alpha = A, P = \{A \rightarrow BBB, B \rightarrow A\}), \tag{5.1}$$

$$\Delta_a = \{\Delta_A, \Delta_B\} \tag{5.2}$$

will generate a sequence of *swarm composition strings* A, BBB, AAA, $BBBBBBBBB$, etc. At each iteration step, either each type-A agent is replicated into three B agents, or agents change from type B to type A. If A agents have no separation urge ($c_1 = 0$), and B-type agents do separate ($c_1 = 1.0$), the generated swarm of agents creates a tree-like structure as in Figure 5.2(a). Note that here and in the following examples we assume $\theta = 1$, that is a matching rule is always applied.

In particular, Figure 5.2(a) displays 243 agents—which are visualized as pyramidal shapes at the branch tips. Both occurring agent types A and B have an upward urge. Since B-agents repel from each other, a bushy crown emerges. Figure 5.2(b) shows a similar set of swarm grammar agents that is forced to climb up a wall, which they cannot penetrate. Once the agents reach to the top of the wall, they are drawn towards a fixed point above and behind the wall. The small flock of agents is visible just ahead of the top branches. In Figure 5.2(c) agents are attracted towards a rotating 'sun' object, which makes them follow a spiral during their upward path. The structure on the right is constructed by a single agent, whereas the left structure involves 20 agents which are repelling from each other.

Each step of applying the production rules (in parallel) represents a decision point for all agents within the system. Contrary to L-systems [338], where only a single 'turtle' is used to interpret a string, we employ a swarm of interacting agents. We do not need to add navigational commands for the turtles within the grammar strings, because the swarm agents navigate by themselves, determined by the agent specifications as part of the SG system. More detailed examples of swarm grammar rewriting that demonstrate further application aspects are given in [343].

Interactive Exploration of Swarm Grammar Spaces

Combining swarm systems with evolutionary computing has to our knowledge only been considered in the context of particle swarm optimization (e.g., [372, 373]) and in swarm-based music generating systems (e.g. [374, 155]). Emergence of collective behavior has been investigated for agents within a three-dimensional, static world [375], but this did not involve interactive evolution. Our *Genetic Swarm Grammar Programming* (GSGP) approach incorporates both interactive, user-guided evolution as well as the utilization of emergent properties from interactions of a large number of agents.

The rewrite rules and agent parameters are represented as symbolic expressions, so that GP can be used to evolve both the set of rules as well as any agent attributes. This follows our framework for evolutionary programming, *EVOLVICA* [365], where all rewrite rules and agent parameters are encoded as symbolic expressions [376]. For the examples we present here, only context-free rules with a maximum string length of three ($|s| = 3$) are applied. We allow at most five rules and up to three different types of swarm individuals per SG-genotype.

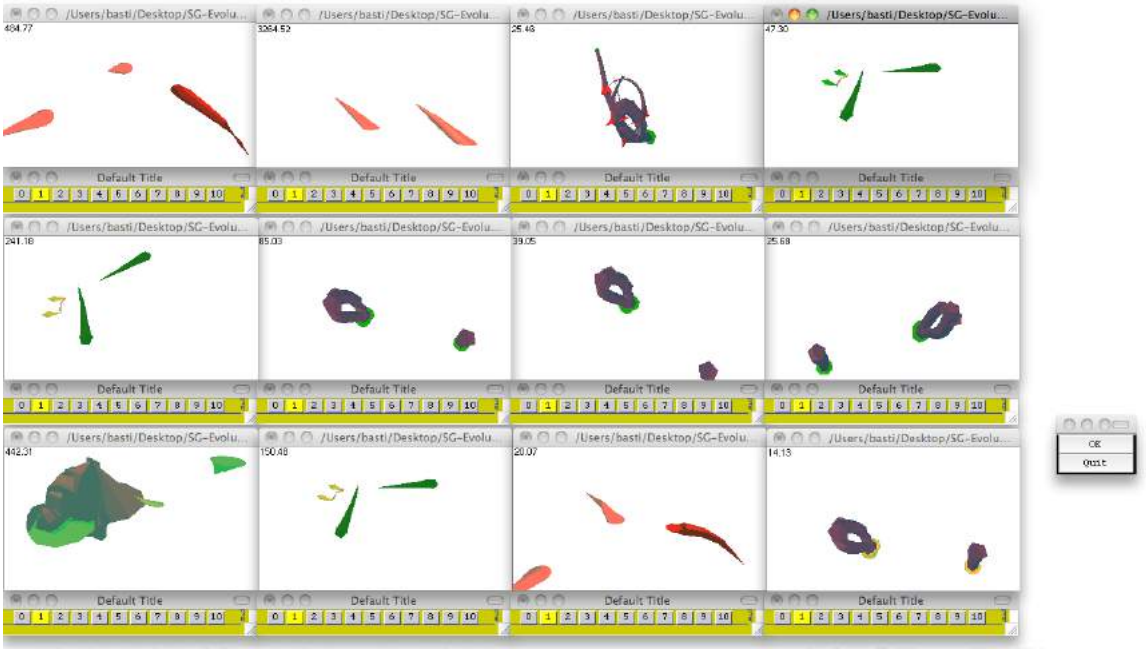
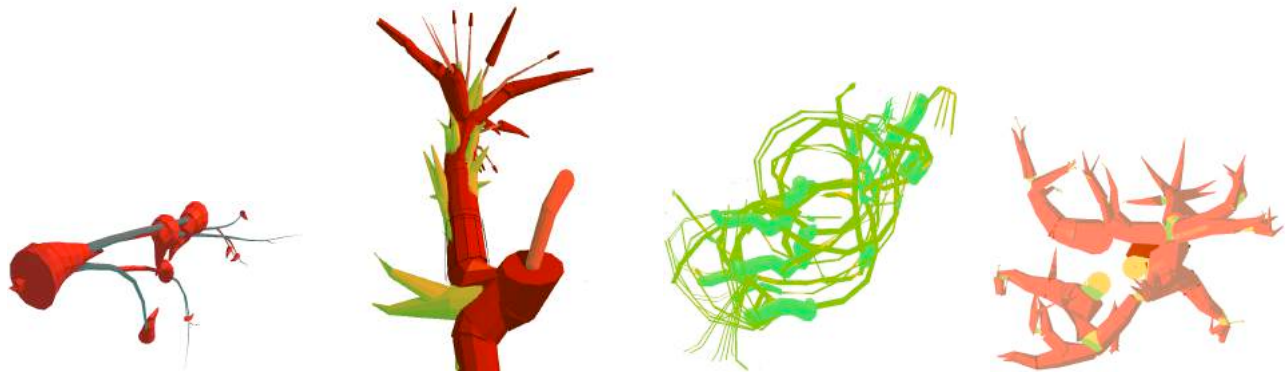


Figure 5.3: Screenshot of the Inspirica GUI that enables interactive evolution based on Mathematica in combination with its genetic programming extension *EVOLVICA*.

In our evolutionary swarm grammar experiments standard GP tree-crossover and subtree mutations are the only applied genetic operators [365]. We use an extension of Inspirica [376], one of our interactive evolutionary design tools, to explore the potential of the described swarm grammar systems.

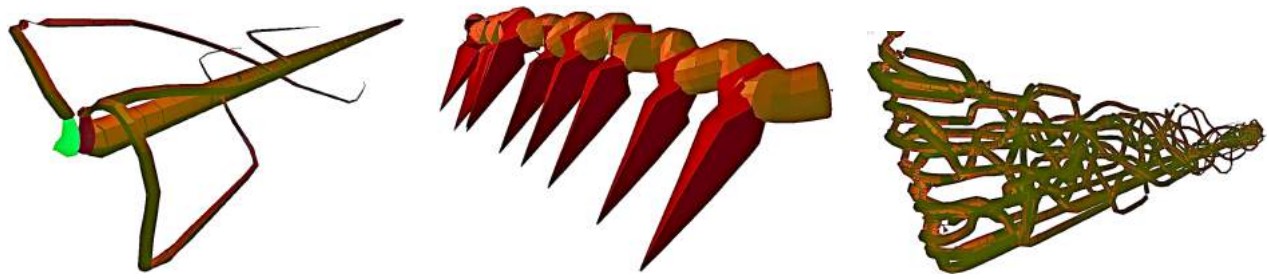
Figure 5.3 displays a screenshot of the *Inspirica* [376] user interface that helps to interactively evolve swarm grammars. All windows display the construction process as it occurs. All designs are true objects in 3D space, hence can be rotated, zoomed and inspected in various ways. After assessment of the presented (twelve) structures, the swarm designer assigns fitness values between 0 and 10 to each solution, and proceeds to the next generation. By means of this approach, one can easily—within only a few generations—create structures as illustrated in Figure 5.4.

The impact of the inter-breeding process, accomplished through crossovers of the SL-system grammars and their associated agent parameters, is illustrated in Figure 5.5. The replication of an agent (as determined by the grammar) and its associated constructions cease as soon as a swarm agent runs out of energy. Since the energy level of an agent is linked to the radius of the built cylindrical shape, the structures tend to look like naturally grown, with smaller tips at the ends. If the agents' energy loss, I_e, is very low, however, the radii of the cylindrical objects



(a) (b) (c) (d)

Figure 5.4: Examples of Evolved Swarm Grammar Phenotypes: (a) Pointy yet smooth nodes connect with long thin branches. (b) A flower-like structure created by a single mutation. (c) Spinning and whirling groups of swarm agents create a woven 3D pattern. (d) An organismic structure with growing tips.



(a) (b) (c)

Figure 5.5: Examples of the impact of interactive breeding: (a) and (b) show two phenotypes that were interbred and whose offspring (c) successfully acquired characteristics of both parent structures. Investigation of the genotypes confirms that a recombinational transfer of a recursively applicable grammatical rule leads to the complex mesh structure in (c).

hardly decrease. Since the energy level is one possible termination criterion, constructions that keep their radii approximately constant often appear in tandem with vivid growth. These effects are illustrated in contents/PART2-02-Development/Figures 5.2 and 5.4.

5.3.2 SG Individuals with Complete Genotypes

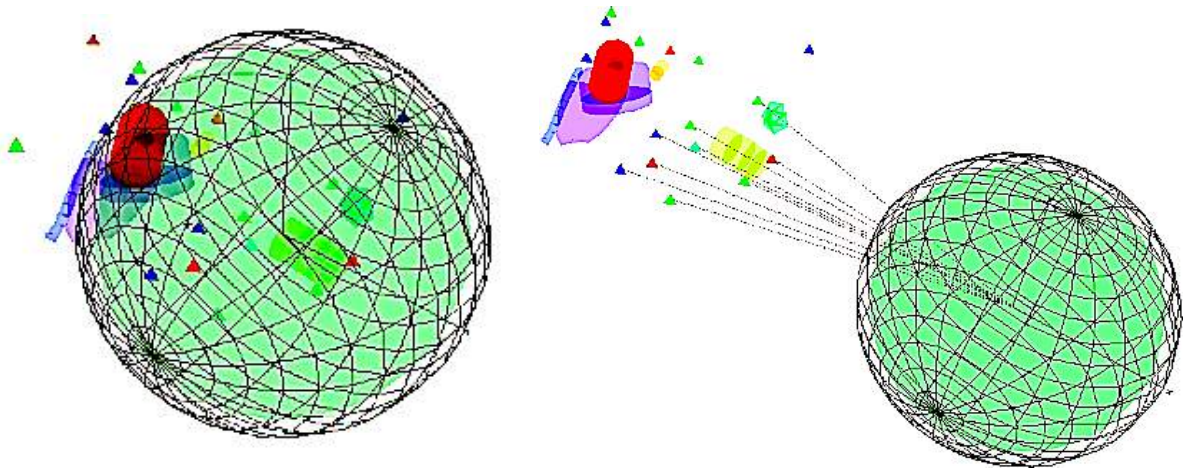
In the previous examples (section 5.3.1) swarm grammars are simulated within separate spaces. In an immersive design ecology, however, one could grow large numbers of swarm grammar structures in a co-existing and co-evolutionary fashion. The encountered phenotypes can then result from massive interactions of heterogeneous swarms. For this to happen, each swarm

agent has to carry the complete genetic information of a swarm grammar $SG = (SL, \Delta)$—this also allows for realtime mutations and crossbreeding of specimens in the virtual environment. This extension of SGs is not unlike in multicellular organisms, where the complete genetic information is passed from parent to daughter cells, and where the differentiation of a cell is performed through reading and expressing specific genetic information.

Spatial Breeding Operators

Our immersive user interface integrates two aspects: visual representation and intuitive manipulation by an external breeder or designer. The latter mechanism is realized by the already mentioned spatial breeding operators, or breeder volumes. Figure 5.6 shows a breeder volume that encloses several swarm grammar agents. Swarm agents that pass through a volume (a sphere in this case) can be influenced in various ways. We use breeder volumes for the crossover and mutation operators, for moving and copying swarm agents, and for boosting their energy levels. Analogous to the watering of plants in a garden, fitness evaluations are only given implicitly by providing more energy to selected groups of agents. In order to facilitate the selective evolutionary intervention, breeder volumes can be placed at fixed positions to perform operations on temporary visitors with predefined frequencies. Additional visuals allow to keep track of previous agent selections. Figure 5.6(b) depicts how previously enclosed agents remain associated with the according breeder volume. This relationship is visualized by the connecting lines.

The visualization interface enables the moving, rotating, and zooming of the camera, or the saving and restoring of specific views and scenario settings (Fig. 5.7). Most of these procedures are already incorporated in the agent software environment *BREVE* which we use as our display and simulation engine [375]. In addition to aspects of visualization, the supervising breeder is equipped with tools to select, group, copy, and move swarm grammar agents. This enables the breeder (designer) to influence the course of evolution within the emerging scenario. The set of possible manipulations also includes mutation and crossover operators to manually trigger changes of the genotypes that encode the swarm grammar rules and the agent parameters.



(a) (b)

Figure 5.6: (a) By means of volumetric tools the immersed breeder can manually select and tinker with the present specimens. (b) Visual cues such as connecting specimens to breeder volumes via dashed lines allow the breeder to keep track of sets of selected agents.

The Swarm Grammar Gardener

Figure 5.7 illustrates how a breeder can influence the emerging building processes within a simple ecology of swarms. In Figure 5.7(a) two swarm agents have built a cylindrical structure with a small side branch. Both agents, which have run out of energy, are still visible at the top left and to the right of this construction. In the next step (Fig. 5.7(b)) a breeder sphere is introduced so that it encloses the agent on the right. Through a contextual menu, this agent is 'revived' by replenishing its energy reservoir. Subsequently, the agent resumes its building process, generates an additional side branch and extends the overall structure further to the right (Fig. 5.7(c)). A similar procedure is applied to the agent on the left. It is captured by the breeder sphere and triggered to first replicate, i.e., make copies of itself, and then resume construction (Fig. 5.7(d,e)). This generates further expansions of the structures and—after further energy boosts (Fig. 5.7(f))—results in the structure depicted in Figure 5.7(g). The pattern continues to grow until the agents run out of energy again.

This is only an example of how external manipulation by a breeder, the ' gardener', can influence the agent behaviors, the building or developmental processes. Their evolution as agents can change their respective control parameters during replication. Agents of a specific type share a swarm grammar, but agent groups can be copied as well, so that they inherit a new copy of their own swarm grammar, which may also evolve over time. This can be accomplished either automatically or through direct influence from the gardener. Figure 5.8 gives a few examples

of evolved swarm grammar ecologies and extracted structures at different stages during their evolution.

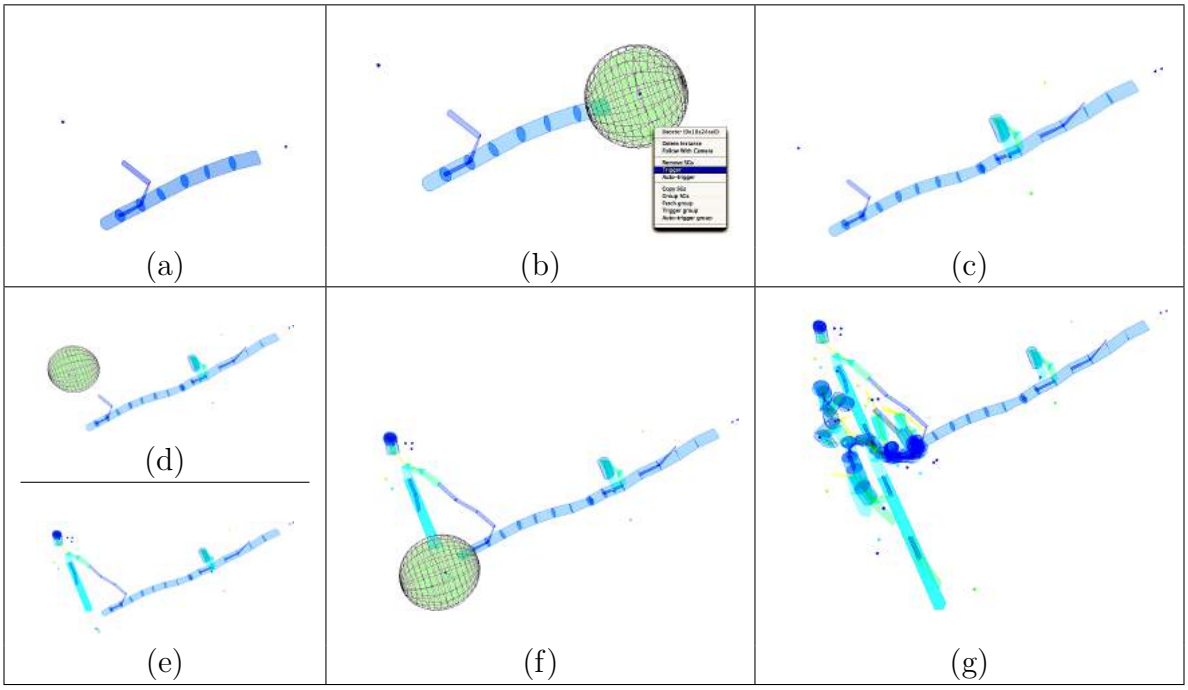


Figure 5.7: Illustration of Interactive Manipulation of Swarm Grammar Agents by an External Breeder. (a) Two agents create an initial structure. (b) A breeder sphere locally infuses energy. (c) Further growth is initiated by the additional energy. (d-e) Replication of an agent triggers further parallel construction. (f-g) Expansion of the structure is continued after another energy influx.

Swarm Constructions in the Arts

In the examples above, the aesthetic judgement of a breeder drove the evolution of swarm grammars. This works particularly well when an artist searches for innovative expressions of certain artistic themes. Swarm grammar constructions are special in that the dynamics of their construction processes are captured within the emerging structures. Local interactions determine the placement of construction elements and the flight formations of the swarm. Inherent in any swarm system, the agents' actions and reactions result in a feedback loop of interdependencies [117]. The diagram in Figure 5.9 hints at the complex relationships that arise in boid systems [1]. Here we don't even consider indirect communication beyond the ever changing neighbourhood relations between the swarm individuals: A swarm agent i perceives a set of neighbours that determine its acceleration. Its changed location, in turn, affects those swarm mates that perceive i as a neighbour. The emerging dynamics are captured in structures that

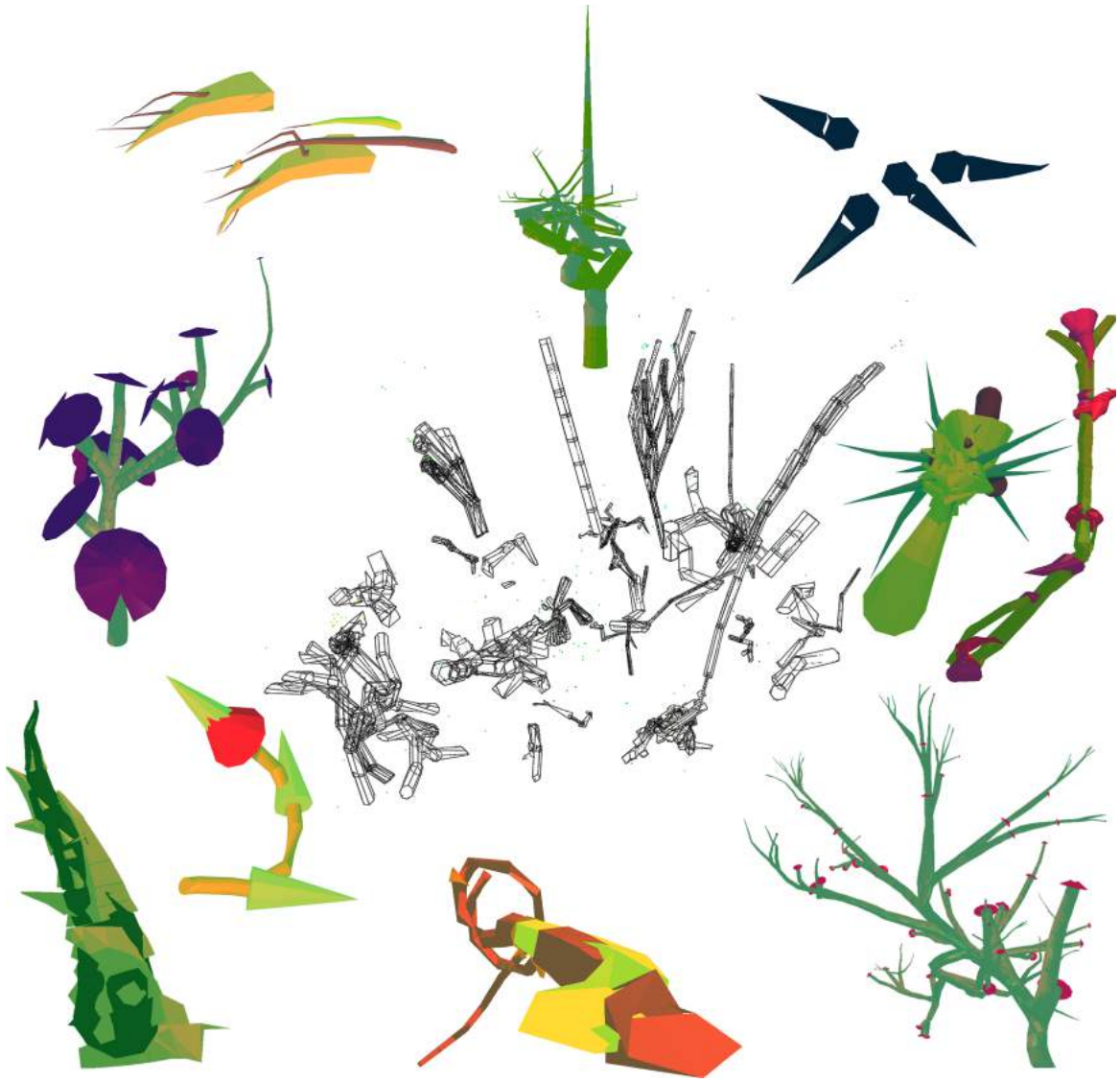


Figure 5.8: Collage of Designs Generated by Swarm Grammars. The figure in the centre illustrates a swarm grammar garden ecology, within which the surrounding designs were created.

exhibit liveliness and spontaneity, contrasting themes, rhythmic movements, tension, organic looks, and rigid forms.

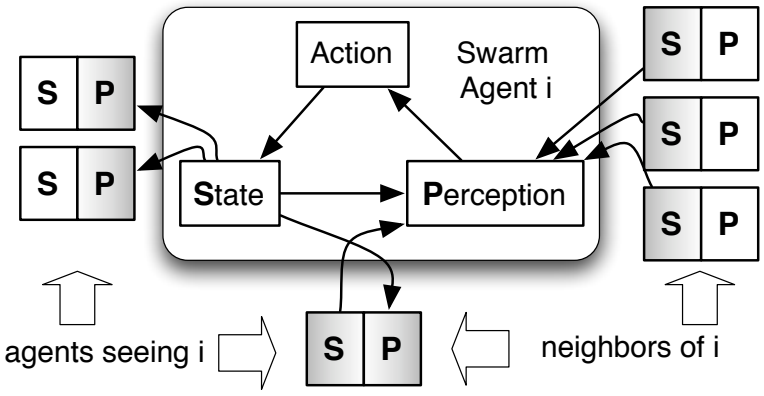


Figure 5.9: The black arrows in the upper box show the direction of influence between perception, action and state of a swarm agent i. The S-P tuples stand for the state and perception modules of other agents that interact with agent i.

Consequently, the artistic interpretation of SG structures can support artistic work in several ways. For example, we composed pieces of computer-generated SG structures and traditionally painted motives [349] and looked at inspiring themes and concepts of artistic works as a whole [347]. Within these explorations—inspired by the architectural potential of swarm grammars—the artist (S.v.M.) combined a collection of swarm structures to create surreal, artificial worlds (Figure 5.10 (a) and (b)). In about 40 interactive evolutionary experiments, the artist bred the sets of swarm grammar structures displayed in Figure 5.10 (c) and (d).

During the evolutionary runs, the artist followed two main objectives. First, robust looking beams should emerge that form a structural mesh, thus opening vast spaces. Secondly, fuzziness, continuity and the resemblance to organic forms should warrant the authenticity of the generated virtual worlds. The colour gradients in the backgrounds emphasize the wholesome, fluent structural architecture in Figure 5.10(a) and the liveliness and dynamics caught in the erratic structures of Figure 5.10(b) with warm and cold colour palettes, respectively.

5.3.3 Rule-based Swarm Grammars

As a second generalization of our SGs, we wanted to go beyond fixed parameters, such as the regularly timed application of reproduction rules or the continuous placement of construction elements. SG rules are now expanded by conditions that each individual agent would relate to,

(a) (b) (c) (d)

Figure 5.10: Diptych of the two pieces (a) *caméléon* and (b) *bighorn sheep*. Acrylic medium on canvas, 23" x 38". Selections of swarm grammar structures bred for the diptych are displayed in (c) and (d), respectively. (S.v.M., 2008)

e.g., specific internal states or perception events. An example of such a rule-based genotype is illustrated in Figure 5.11.

Breeding Architecture

Perception-induced rule execution allows for indirect, so-called *stigmergic* communication by which social insects, such as ants, termites and some wasp and bee species, are assumed to coordinate large parts of their construction behaviours [377, 229, 230]. Stigmergy can then be harnessed to create assortments of innovative architectural SG designs by means of computational evolution [345, 348]. In order to automatically drive the evolutionary processes, we need a way to assign fitnesses to SG specimens. There are several aspects that should be taken into account when it comes to the measuring of structural complexity, as we have outlined in section 5.2.2.

In addition to the analysis of the genotype of a swarm grammar, two aspects can be incorporated into the fitness assignment of an evolutionary algorithm: (1) the construction processes and (2) the emerging structures. Structural analysis is either very course grained, considering for example the overall volume and the proportions, or computationally very costly, for instance when attempting to identify hierarchies and re-occurring modules. Therefore, we put an emphasis on the observation and classification of the construction processes.

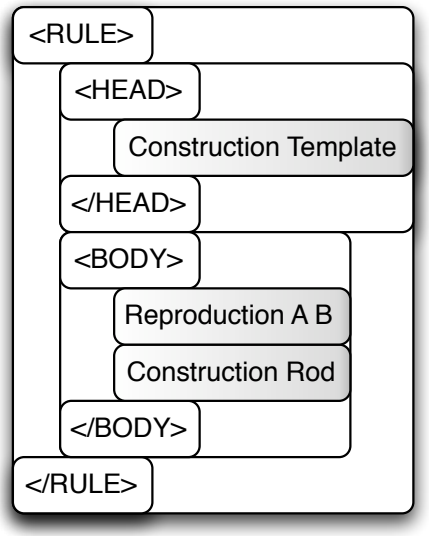
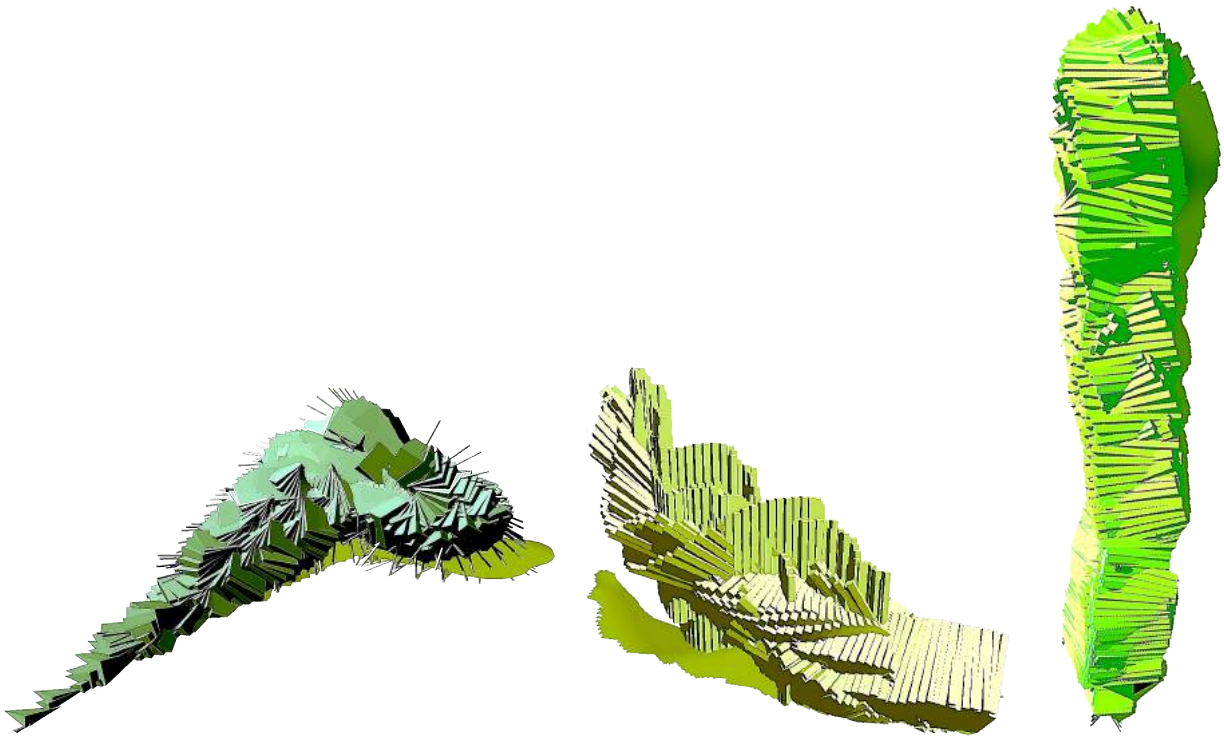


Figure 5.11: An example of a behavioural rule of a swarm grammar agent. Instead of continuous construction and regularly timed reproduction, this rule triggers the reproduction of two agents (types A and B) and the construction of a rod whenever the acting individual perceives a construction template.

In particular, in a series of SG breeding experiments for architectural design, we promote productivity, diversity and collaboration, and we prevent computational outgrowth of the generated structure. Our detailed evolutionary approach to automatic SG evolution is presented in [345]. In order to reward productivity, the SG constructions are compared with (simple) pre-defined structures. More specifically, construction elements built inside a pre-defined cubic shape contribute positively to an SG's fitness, whereas constructions outside the cube decrease it. This is similar to an approach we used in [378]. Diversity is traced as the total number of expressed agent genotypes, as well as the number of deployed construction materials or construction mechanisms. In order to foster collaboration between the SG agents, we observe the numbers of perceived neighbours averaged over all active agents and over time. Low values of perceived neighbours imply that no direct interactions are taking place, whereas large values mean that the agents are trapped within small spaces. Randomly initialized swarm grammar systems can quickly exhaust the provided computing power: Fast, possibly unconditional sequences of SG rule applications may result in exponential agent reproduction. Temporarily, such explosions of activity could be beneficial, for example in designs that produce large numbers of ramifications. In the long run, however, such an overwhelming demand on computing requirements has to be avoided. As a simple means to prevent prohibitive outgrowth, yet allowing for temporary leaps of activity, we set a time limit for the computation of one specimen. Thus, we filter inefficient SGs during the evolutionary experiments.

Three examples of architectural SG models are displayed in Figure 5.12. The flowing and organic shapes built by the bio-inspired, generative SG representation promise to support the design efforts of architects [348, 379]. Utilizing the extended swarm grammar model to breed architectural designs is not only interesting from a creative and innovative perspective on aesthetics, but it also bears the potential for optimizing architectural designs in respect to ecological and economical aspects. Such ecological criteria could be temperature regulation and ventilation [380], adaptation of building structures to the surrounding landscape, utilization of sun exposed structures for electric power generation, and other evolvable and measurable features [381].



(a) (b) (c)

Figure 5.12: The displayed architectures are the result of automated evolutionary computation processes. They emphasize the dynamics of the swarm-driven construction process.

Driving Evolution with *EvoShelf*

In order to further the design and the analysis of evolutionary design, we have developed *EvoShelf* [382], a reliable storage/retrieval system for computational evolutionary experiments and a fast browser for genotype and phenotype visualization and evaluation (Figure 5.13). Through *EvoShelf* we have been able to discover that our preliminary breeding experiments

(section 5.3.3) tend to produce over-fitting SG populations and predominantly promote the variety of deployed construction elements.

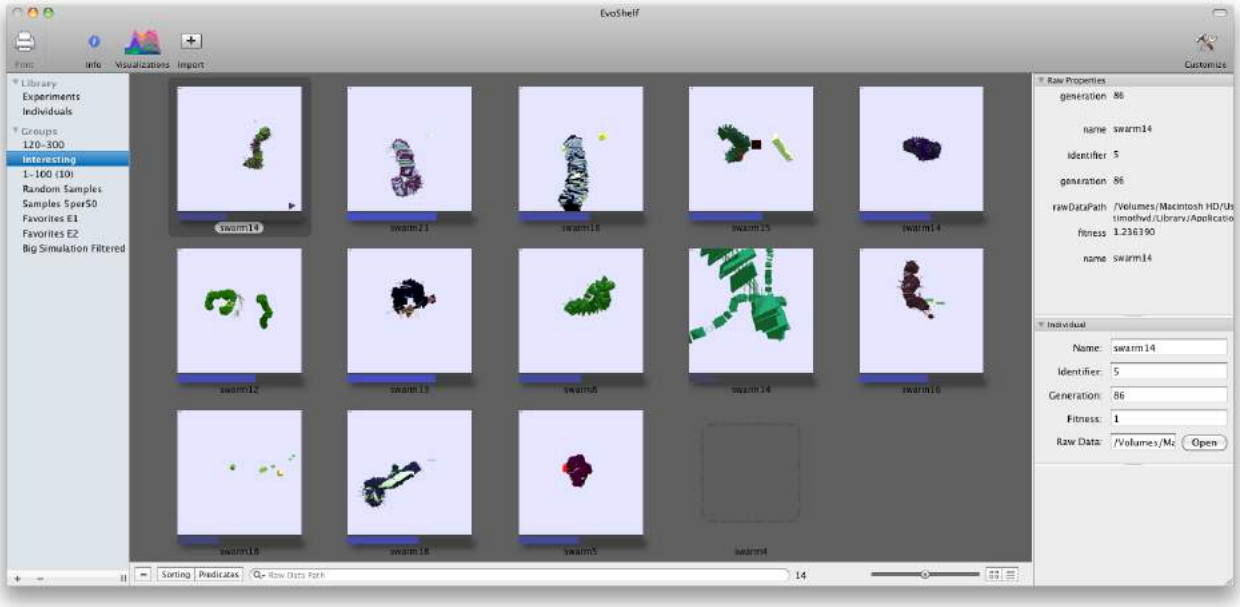
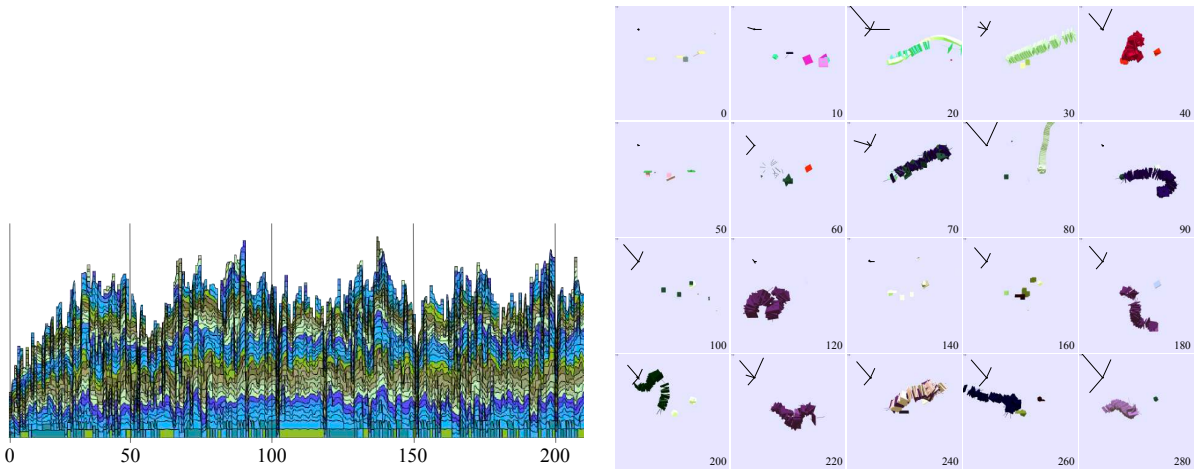


Figure 5.13: We were able to improve our architectural SG experiments by means of our management and analysis software *EvoShelf*.

Figure 5.14(a) depicts a corresponding, representative *FitnessRiver* plot that was created by *EvoShelf*. Our *FitnessRiver* visualization method stacks the fitness values of individuals on top of each other. The fitness of an individual is proportional to the width of its current. Different colours are used to distinguish between successive individuals. Discontinuing currents indicate the removal of an individual from the evolutionary process. In the *FitnessRiver* visualization the x-axis represents the sequence of generations. The shown plot exposes stagnating and fluctuating fitness development after about 100 generations. A bias towards specific construction materials deployed by the SG specimens could be identified in star plots representing phenotypic features as seen in the corners of the SG visualizations in Figure 5.14(b). Based on these investigations we are able to adjust our fitness functions and the configuration of our breeding experiments [382].

5.3.4 A Streamlined, Accessible Swarm Simulation Framework

Rule-based swarm systems seem to be a good fit to capture biological models [304, 383, 289, 384]. However, there are several hurdles that make it hard to deploy swarm models in fields outside



(a) (b)

Figure 5.14: *EvoShelf* provides the user with quick visualization methods for global fitness trends and local comparisons, as in (a) the *FitnessRiver* plot and (b) star plots of the specimens' features, respectively.

of computer science:

1. The predicates and actions that drive the simulations—e.g. the detection of a chemical signal or the deposition of a particle—depend on the modelling domains and usually have to be re-implemented for different experiments. Still, many of these operations can be abstracted, parametrically adjusted and reused in different contexts. The integration of these operations into a rule-based formalism also makes it possible to utilize functionality from various computational engines such as physics engines or general differential equation solvers within a single modelling framework.

2. Depending on the degree of specificity of a rule's condition and its associated actions, a theoretically simple interaction can result in an over-complicated representation. A graphical description of the predicates and the associated actions can amend this issue.

3. As swarm simulations often exhibit complex behaviours, little details—for example the order of execution and the discretization steps in a simulation—can greatly influence the outcome. Therefore, we think it is crucial to design models based on a unified algorithmic scheme.

We have devised *swarm graph grammars* (SGGs) to alleviate some of the challenges discussed above [346]. SGGs provide a graphical, rule-based description language to specify swarm agents

and a generalized algorithmic framework for the simulation of complex systems. Fundamental operations such as the creation or deletion of programmatic objects, as provided by formal grammars, are part of the SGG syntax. Through SGGs we can capture (metabolic) functions at multiple biological scales. We can capture processes of secretion and diffusion [301] as well as consumption/removal and production/construction [385]. As a consequence of the graph-based syntax, SGGs capture the simulation state in a global graph at each computational step. Thereby, the continuous re-shaping of an interaction topology of a dynamic system is traced and interdependencies that emerge over the course of a simulation can be represented graphically.

SGG Rule Description

An SGG agent's behaviour is described by a set of rules (Figure 5.15). Each rule tests a set of predicates (solid edges on the left-hand side) and executes a set of actions (dashed edges on the right-hand side) in respect to the acting agent itself (reference node) or other agents. Nodes represent individual agents or sets of agents. In Figure 5.15, the acting agent is displayed as an orange node with a black border. Other agents or agent groups are depicted as grey nodes. The application of the rule is associated with a frequency and a probability. Sets of predicates can attempt to identify an arbitrary number of agents. The relative location, i.e. the two-dimensional coordinates, of the node on the left-hand side of the rule is matched with its appearance on the right-hand side of the rule. If a node does not reappear on the right-hand side, it implies that its corresponding agent has been removed. If a node appears at a location that is unoccupied on the left-hand side, a new node is created. Figure 5.15 shows an example rule. This rule is applied with a probability of $p = 0.3$ at every fourth time step ($\Delta t = 4$) of the agent simulation. One (arbitrarily chosen) node that fulfills *predicateX* and *predicateY* is affected by *actionJ* and *actionK*. Also note that a new node is created and is initialized, for which no reference had existed before. In case there are at least 6 nodes that fulfill *predicateZ*, they will all be removed.

Swarm-based Embryogeny & Morphological Development

Swarm graph grammars enable us to closely collaborate with researchers from other disciplines such as architecture, biology or medical sciences. Following the footsteps of previous works in

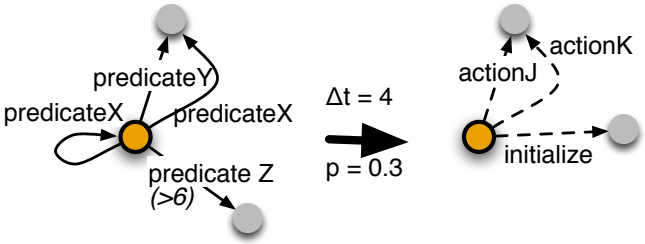


Figure 5.15: An SGG rule that queries the reference node itself (orange), other individuals (grey) and sets of interaction candidates. The consequence of the rule defines the interactions, such as deletion of nodes and initialization of a new node.

artificial embryogeny and morphogenetic engineering [386, 387, 388, 389, 390], we have begun to investigate simulations of biological developmental processes. In a series of (at this point naive) experiments we integrated high-level SGG agent behaviours (maturation and proliferation) with physical mechanics (collision and impulse resolution) (Figure 5.16). Tissue cells (blue: not mature; red: mature) within the vicinity of a signalling molecule (green) start proliferating. Collision resolution through an embedded physics engine allows the cells to assemble[2]. The emerging protuberance is slanted to the right in accordance with the initial distribution of signalling molecules.

Initially, we were surprised to see that the protuberance in Figure 5.16 turned out symmetrical, despite its one-sided development. We speculated that this due to a lack of simulated cell polarization. However, after a series of systematic simulations, we found out that the effects of polarization would, in combination with proliferation, still be overturned by the physics interactions and again result in spherically distributed, aggregated cells (Figure 5.17).

Using this same agent-based approach, we have begun tracing embryogenic developments in mice. Volumetric embryo data provides a basis to populate initial tissue layers with cells (Figure 5.18). Basic intra- and inter-cellular interactions will then dictate the shaping of existing and the creation of new tissue layers as shown in the mesh-deformation in Figure 5.19. Here sentient swarm agents play the role of vertices on a graphical surface. In this context, dynamic mesh generation and manipulation could become part of the agents' sets of possible actions [391].

[2]In the given experiment we rely on the Bullet physics engine, http://bulletphysics.org

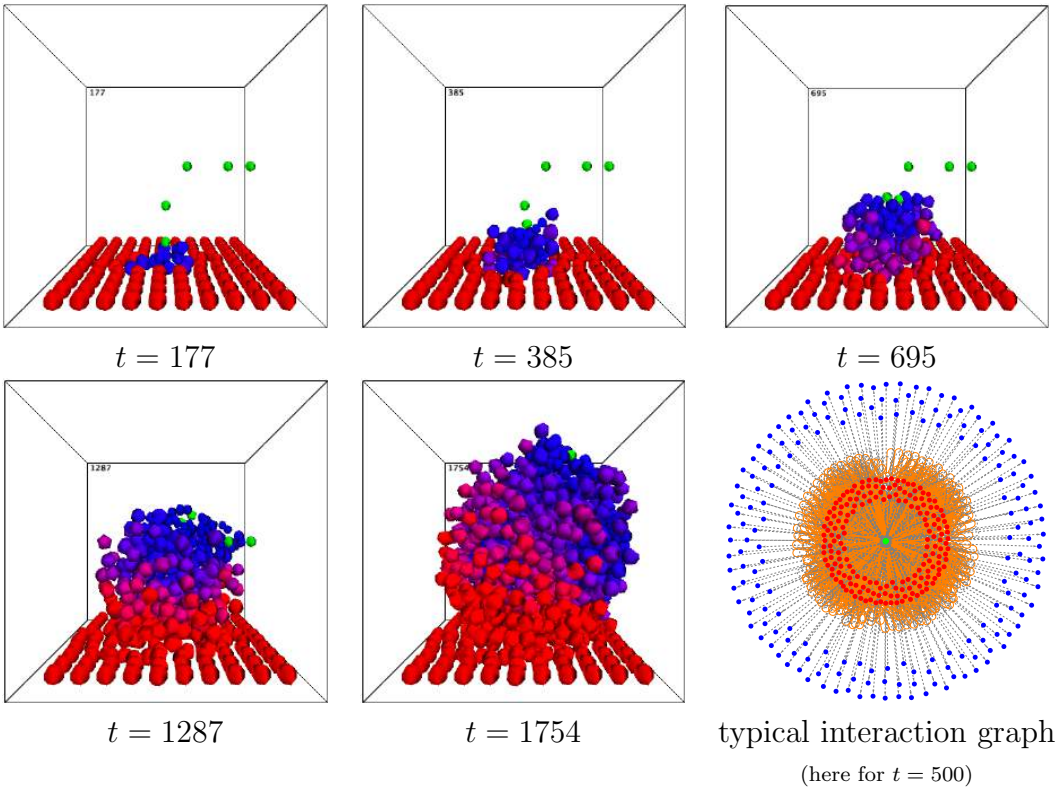


Figure 5.16: The proliferation of mature cells (blue: premature; red: mature) is dependent on the proximity to growth factors (green). At any time of the simulation, large numbers of agents are informed by growth factors leading to typically dense but homogeneous graphs that reflect their interactions.

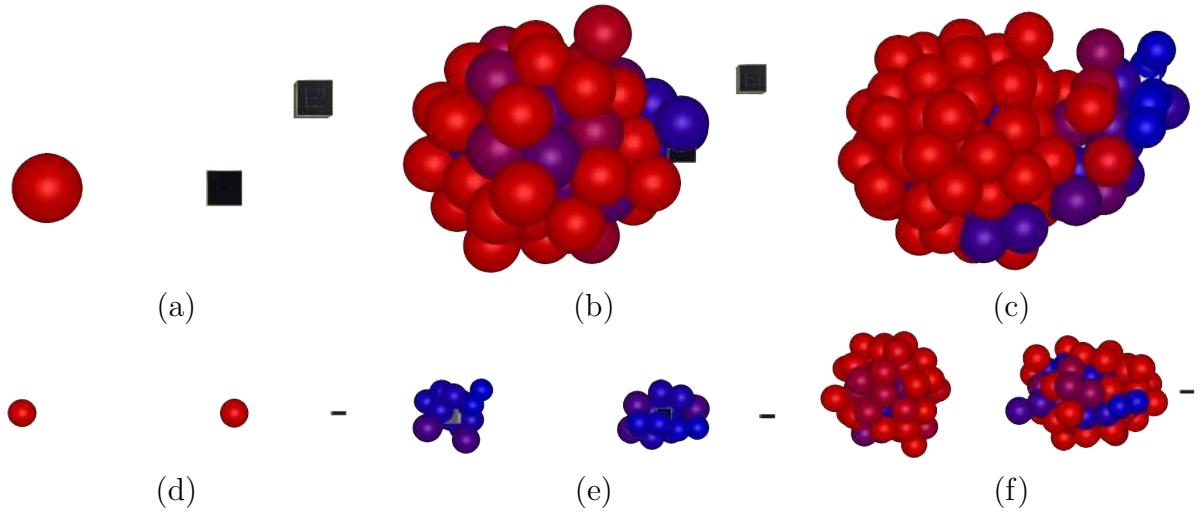
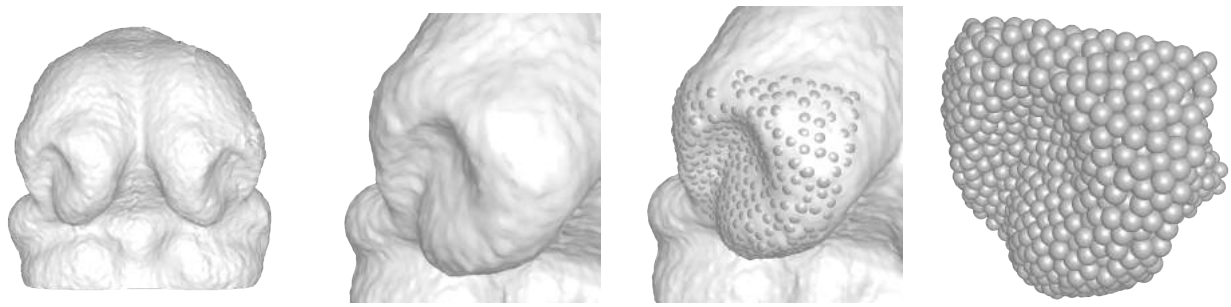
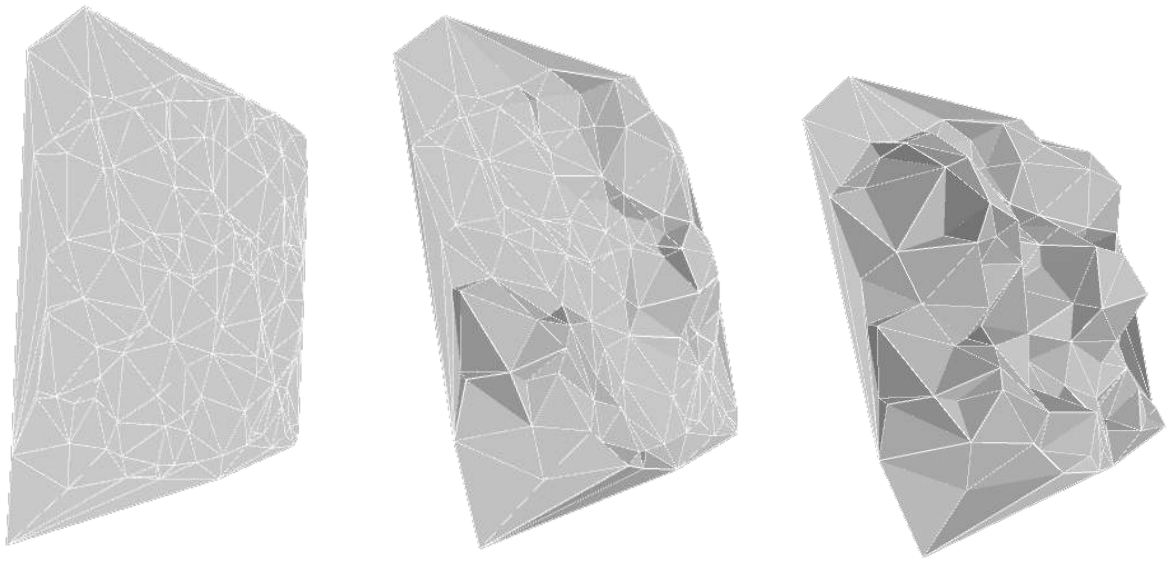


Figure 5.17: (a)-(c) show the same proliferation process as in Figure 5.16 but with only one initial cell (growth factors are illustrated as black cubes); (d)-(e) show two simultaneously growing protuberances, whereas the cells on the right-hand side obtain a polarization aligned towards the polarization signal to the right (black box).



(a) (b) (c) (d)

Figure 5.18: (a) We start with a volumetric scan of a mouse embryo, (b) zoom into the region of interest and (c)-(d) populate it with swarm agents.



(a) (b) (c)

Figure 5.19: Swarm agents occupy the vertices of a three-dimensional surface which is deformed based on their interactions and movements.

5.4 Summary and Conclusion

Inspired by the construction abilities of social insects, we started investigations into virtual constructive swarms [343]. We designed swarm grammars (SGs) as a computational developmental representation that combines the ideas of artificial swarm simulations[3] with the compositional regulation expressed by formal grammars [339]. L-systems are a prominent approach to translate formal grammars (rewrite rules) into the realm of developmental models [392] (section

[3]Artificial swarms can be considered a special case of agent-based modelling with a focus on large numbers of locally interacting individuals and the potential of emergent phenomena which cannot be inferred from the individuals' abilities.

5.3). SGs allow for completely unrestrained interaction topologies and provide a simple way to integrate interactions beyond population control and fixed local neighbourhood relationships, which represents an expansion from the more constrained L-systems.

In an iterative process of unification and extension of the initial swarm grammar representation, we first incorporated a complete swarm grammar genotype into each swarm agent (section 5.3.2), then started describing its behaviour as a set of perception-reaction rules (section 5.3.3). The original idea of using grammatical production to determine the composition of the swarm population became part of a more generic agent-based representation [54, 55]. To even further the modelling capacity of swarm-based simulations, we designed swarm graph grammars (SGGs) as a means to graphically represent swarm agent interactions and to explicitly model inter-agent relationships that might influence the dynamics of the simulations (section 5.3.4). Swarm graph grammars provide a modelling language that can be used for interdisciplinary investigations. In a collaborative project, we have begun tracing complex developmental processes in mice (section 5.3.4).

An expanded degree of freedom in SG representations required systematic exploration of configuration spaces. We addressed this challenge by means of computational evolution [343, 344, 345]. In particular, we relied on interactive evolution to explore structural spaces (section 5.3.1) that inspired artistic works [347, 349] (section 5.3.2). We furthered this approach by the possibility to breed large swarm grammar ecologies in virtual spaces (sections 5.3.2 and 5.3.2). We promoted structural complexity by considering the frequency and diversity of interaction processes among swarm agents in order to generate interesting architectural designs (section 5.3.3). More systematic investigations in accordance with scalable complexity measures as outlined in section 5.2.2 might yield a better performance in the context of breeding innovative designs.

With the evolution and exploration of swarm grammars, we have been building methodologies and toolkits that support modelling and simulation of developmental systems in a multitude of domains. Evolutionary computation techniques enable us to find swarm system configurations to trace more or less desired or innovative outcomes for artistic or scientific simulations. However, we are aware that there are several major obstacles to be addressed before our methodologies can become instrumental for broad application. Currently, we focus on two issues. First, we attempt to reduce the computational complexity that inevitably arises from offering a very

generous and expressive representation [9]. Second, we are constantly engaged in improving the usability and accessibility of our modelling representation itself.

Chapter 6

A Trans-Disciplinary Program for Biomimetic Computing and Architectural Design

In this article, we present the conceptual foundation and selected studio works of two iterations of our trans-disciplinary university course program that integrates biomimetic computing and architectural design for graduate students in Architecture. In particular, we first present the motivation behind and the implementation details of a basic framework for self-organizing multi-agent computer simulations. Second, we highlight its conceptual presentation to the students as well as its appropriation by the students through examples of the students programmatic and material implementations in architectural design projects.

Sebastian von Mammen, Joshua M. Taron: *A Trans-Disciplinary Program for Biomimetic Computing and Architectural Design.* ITcon: Special Issue CAAD and Innovation 17 (2012), pp. 239–257.

6.1 Introduction

Multi-agent systems promote modeling of complex processes by researchers and designers without the need for a profound background in Mathematics. Conceptual models can be directly translated into programming code and the consequences of a previously theorized model can

visually unfold, undergo rigorous analysis, and experience iterative improvement. Agent-based modeling also empowers designers to apply a paradigm of self-organizing systems: swarms of reactive software agents engaging in complex interactions, potentially even reproducing constructive processes. Experiencing and investigating complex systems in nature is another important aspect that promotes the outlined approach to design. Developmental processes in organisms, evolution, self-organizing formations in cell populations or animal societies all serve as an invaluable source for inspiration and for comprehending the ways decentralized, self-organizing, emergent multi-agent models can carry fruits for research and design.

In this article, we briefly stress our approach to teaching and training the ideas of agent-based modeling and related topics around complex biological systems. We will then present some of the programmatic and material project works implemented by the students over the course of two iterations of our program. In order to serve our trans-disciplinary program to the students, we have established a course setup between Computer Science and Architecture. A computer science course on biomimetic computation provides the theoretical foundation and programming know-how for developing agent-based software simulations with a focus on developmental, generative, and interactive processes. Architecture students subsequently or concurrently enroll in an Architecture research studio in which they have the opportunity to apply and evolve their agent-based models developed in the Computer Science course. An inspiring feedback cycle emerges from the trans-disciplinary, theoretically founded and practically applied tandem of project-driven courses.

The contents and the coursework are closely attuned to maximize the opportunity for mutual synergetic fertilization of skills and ideas. In the Computer Science course, students are first familiarized with basic concepts of computational processes and algorithms using the Processing programming environment[1]. Units on coding basics culminate in live programming demos that apply the gathered knowledge about basic data structures and process flow. Simple yet colorful simulation examples are crafted from scratch in front of the class, thoroughly discussed and made available online for future reference[2]. Subsequent lecture units present biological examples of concepts like developmental growth, self-organization and evolutionary processes. Corresponding programming codes are presented in class. Finally, students projects commence, maturing from the initial proposals over prototype implementations into original architectural

[1] `http://processing.org/`

[2] `http://www.vonmammen.org/biocaad2011/material/`

design works (supported through the Architecture research studio).

The Architecture research studio component of the endeavor provides an outlet through which these computational processes can be tested at a variety of architectural scales and formations. By focusing on complex processes in heterogeneous urban environments, multi-agent systems serve as both a tool for mapping cities as well as for the production of architectural design techniques. In our case, students are assigned a partially completed skyscraper in downtown Calgary as their site with the task of intervening in the typical procedural construction/assembly processes necessary to complete the tower. The exercise challenges students to use agent-based models to graft into an already ongoing procedural process thereby augmenting its formal, visual and programmatic performance. The results of the studio are series of new tower iterations using agent-based techniques developed and supported through the Computer Science course.

These agent-based designs provide the material results of the trans-disciplinary exercise, which are evaluated for the purposes of improving the next iteration of the experiment. Methods are discussed that might allow students to improve upon previous years achievements and thus increase the intensity and intelligence of the models themselves over the course of time.

6.2 Programming Nature

Computation happens through manipulating data. Traditionally, sequences of instructions that determine how certain data are manipulated are subsumed under high-level commands, generally referred to as procedures, functions, macros, or methods. Methods can be associated with specific data objects that combine various kinds of information, e.g. symbolic strings, numeric data, or other data objects. A repeatedly occurring example would be a Person object, for instance person1 with the attributes name = Susan, age = 32 and gender = female. The execution of a method in respect to a Person object could, for instance, update Susans age to 33.

Similar to subsuming instruction sequences, objects and their associated methods can be inherited by other object classes. An Employee class, for example, could expand the attributes and methods of the Person object class. This object-oriented programming approach represents the state-of-the-art programming paradigm in software engineering. It is of great value because the

programmer can immediately understand and work with complex code objects and use them for creating his own software. A comprehensive introduction to object-oriented programming is provided by [393].

6.2.1 Agent-Based Programming

Agent-based programming is an extension of the object-oriented approach. It turns passive data objects into active agents that act in accordance with their behavior, their situation and their available data[53]. One speaks of Multi-Agent Systems (MAS), if there are multiple agents at work. The programmer endows the agents with behaviors and properties in such a way that they work efficiently together and accomplish computationally challenging tasks. Potential benefits of MAS can be high robustness as failures in parts of the system can be compensated by intact agents or high efficiency as tasks can be performed in parallel and be assigned with respect to the involved agents specializations.

MAS lend themselves naturally for designing biomimetic computational models, in which systems of molecules, cells, organs, organisms, or societies are retraced. Individuals in these systems act based on their own agenda and contribute to the emergence of high-level processes or designs [230]. The structural properties and the behaviors of living organisms have evolved to yield streamlined, adaptive metabolic processes to occupy and exploit ecological niches. The agent-based modeling approach allows the designer to directly map physiological properties and biological behaviors to computational representations. The only limitations are the knowledge and creativity of the designer on the one hand, and computational power on the other hand.

6.2.2 Swarms

MAS can be designed in many ways. One can, for instance, implement a centralized controller agent that oversees the ongoing processes and concerts the activities of the remaining agents as it sees fit. Inspired by biological systems such as social insect societies, one can alternatively attempt to configure the agents in such a way that a centralized control is not required. A system of such decentralized agents brings a number of advantages: (1) It is generally more robust against failure as there is no crucial, central part that can go missing. (2) The computational

cost for coordinating the agents is reduced by the agents making locally informed decisions. (3) If the task at hand can be divided into independent subtasks, they can be accomplished faster as there are no holdups. Besides such computationally intriguing properties of decentralized systems, there are other aspects that reach even further. For instance, they support the idea of simulating biological self-organization, where a system can reach self-maintaining states independent of its initial configuration [337]. In general, one can say that a decentralized system is a system whose agents can act freely, whereas any kind of control infrastructure introduces varying degrees of limitations in respect to the possible interaction processes. Of course, depending on the system, a rigid control infrastructure might actually be vital, like the coordination of our motor-sensory activity through the central nervous system. Along these lines, we would like to underline that control infrastructures are the results of self-organizing processes themselves. Therefore, decentralized MAS, or swarms, seem to be the least biased, most direct, and thus, most profound approach to computational modeling.

6.2.3 Development

Ultimately, living organisms are biochemical structures that drive their own development, maintenance, and reproduction. These seemingly distinct objectives can all be reduced to systematic metabolic processes, that is the construction and destruction of products ranging from simple molecules to large molecular chains to cells and complex tissues. From this perspective, processes describe the flow of state changes, whereas structures refer to materializations that persist for a perceivably long period of time. Even this careful attempt to distinguish processes from structures emphasizes the role of the observer and it forces us to accept that structure and process are two closely interwoven aspects of life.

Computational swarms can retrace developmental processes, if their interactions yield persistent structures. Swarm agents can create structures in numerous ways. They can, for instance, become part of a larger structure like simulated molecules in Artificial Chemistries [394]. They can deposit building blocks when building their nests like wasps [395], or hollow out tunnels and chambers like ants [377].

Due to the swarm agents degrees of freedom, it is a challenge to assign them behaviors and properties that make them interact in a productive, coordinated fashion [229]. The structures

built by a swarm, however, provide meaningful evidence of the swarms productivity [343], which can serve to find swarm configurations that yield desirable designs, for instance by means of evolutionary computation [345].

6.3 Hands-On Code

We can only assume that a small group of students has been exposed to programming or 3D modeling before enrolling in our trans-disciplinary program. As a result, we have to guide them through the very first steps of a programming curriculum to arrive at the point where they are empowered to read and manipulate programming code or to be motivated to design and implement programs from scratch.

Visual programming environments like Grasshopper/Rhino, Quartz Composer, or Max/MSP provide high-level interfaces that make it easy to compose intriguing programs by hiding implementation details that are usually unimportant for the designer. These environments can offer simple interfaces because they constrain the way designers think, i.e. by forcing them to follow a functional programming paradigm.

Therefore, in addition to the advantages of these visual development environments, we teach the students in Processing [143], an environment that empowers them with the expressiveness of the established, object-oriented Java programming language. Understanding programming in terms of a generic programming language at the level of algorithmic instruction sequences and memory manipulation allows one to naturally understand other languages and high-level interfaces as well. Furthermore, it enables the programmer to break out of an imposed programming paradigm, and in the case of our trans-disciplinary discourse, create the programming infrastructure for agent-based models and simulations.

6.3.1 Surface as Architectural and Mathematical Territory

Architectural form serves as a common territory where both visual and agent-based programming techniques can be deployed. While students are being introduced to algorithmic approaches in the Computer Science course, the Architecture studio runs through a series of 3D

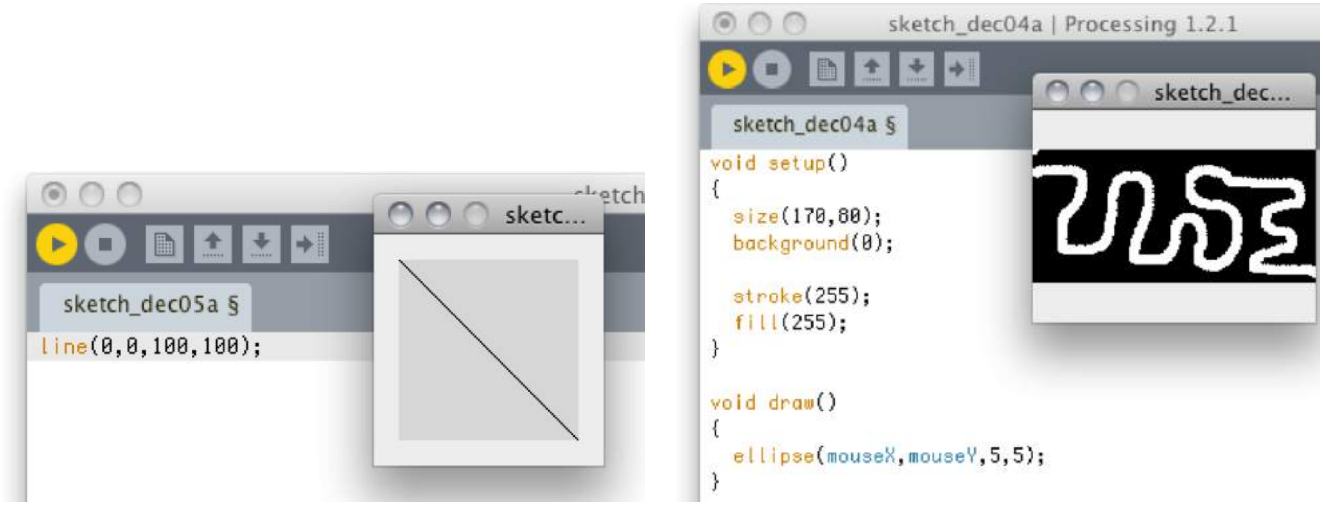
modeling exercises enabling students to tackle architectural problems of scale, massing and circulation while allowing students to become familiar with non-linear geometric relationships within those visual environments such as points, curves, surfaces and manifold spaces (relying on a NURBS geometry representation). The results are designed not just for producing spaces for human inhabitation, but more so for the purpose of defining explicit mathematical territories for the students yet-to-be-developed biomimetic code to inhabit and further articulate.

The exercises continue to evolve in complexity by employing tactics of object instancing, duplication and formal reproduction. This is particularly useful in partially previewing problems and opportunities afforded by MAS. Principles of transformation, gradient change and morphological part-to-part behaviors in these architectural explorations establish programmatic strategies and aesthetic sensibilities that influence and inform students Computer Science projects.

6.3.2 Getting Started with Swarm Programming

Processing is widely used in architecture and art [396, 397, 398]. Writing a program, or sketch in Processing lingo, can be as simple as typing a drawing command such as line(0,0,100,100); into its editor window and clicking the play button (Fig. 6.1(a)). Comprehensive documentation and references are accessible through Processings menu. Fig. 6.1(b) shows a simple interactive Processing sketch that, when started, changes the simulation window size to 170 by 80 pixels, sets its background color to black (color value: 0) and sets the paint color to white (value: 255). For as long as this sketch is running, a circle of radius 5 will be drawn where the mouse pointer hovers over the simulation windowresulting in a squiggly line in the given example.

Fig. 6.2 shows a basic swarm-programming infrastructure in Processing code. In its setup() method, new Agent objects are created. Their locations are set to somewhere on the canvas (dimensions: width x height). The newly created Agent objects are stored in a list called swarm. The draw() method iterates through this swarm list, executes each swarm agents act() method and renders it as a circle on the canvas. In the given example, one agent may react to all the others, i.e. the whole swarm informs each agents actions. As a result, the swarm list is used as the interactionCandidates parameter of the agents act() method. Instead of changing its or its partners state, the agents in the given example only indicate potential interactions by drawing a line to their potential interaction partners. In the given case, all other agents within



(a) (b)

Figure 6.1: (a) Processing offers an easy-to-use editor and various predefined drawing commands such as line(). (b) Code inside the setup() method is executed when the simulation is started. draw() is executed repeatedly until the simulation is stopped.

a distance of 15 units is considered for interaction.

The example shown in Fig. 6.2 has two purposes. First, it shows how a very generic swarm-programming infrastructure can be created. Second, it indicates the potential interactions by drawing lines between subsets of agents. Extensions to the Agent class in respect to its attributes and its act() method infuse the model with meaning. The designer/programmer has to decide what the agents represent, e.g. construction modules, inhabitants, or physical currents, what their relationships are, which control instances and constraints should be applied, and how emerging processes and structures inform each other.

6.4 Explorations of Biomimetic Design: Iteration 1 (2010)

The students in our program are asked to develop a sense for dynamic swarm systems and explore how they can impact the creation of architecture. In this section, we identify continuing trends across two years of studios that have emerged through the application of biomimetic code.

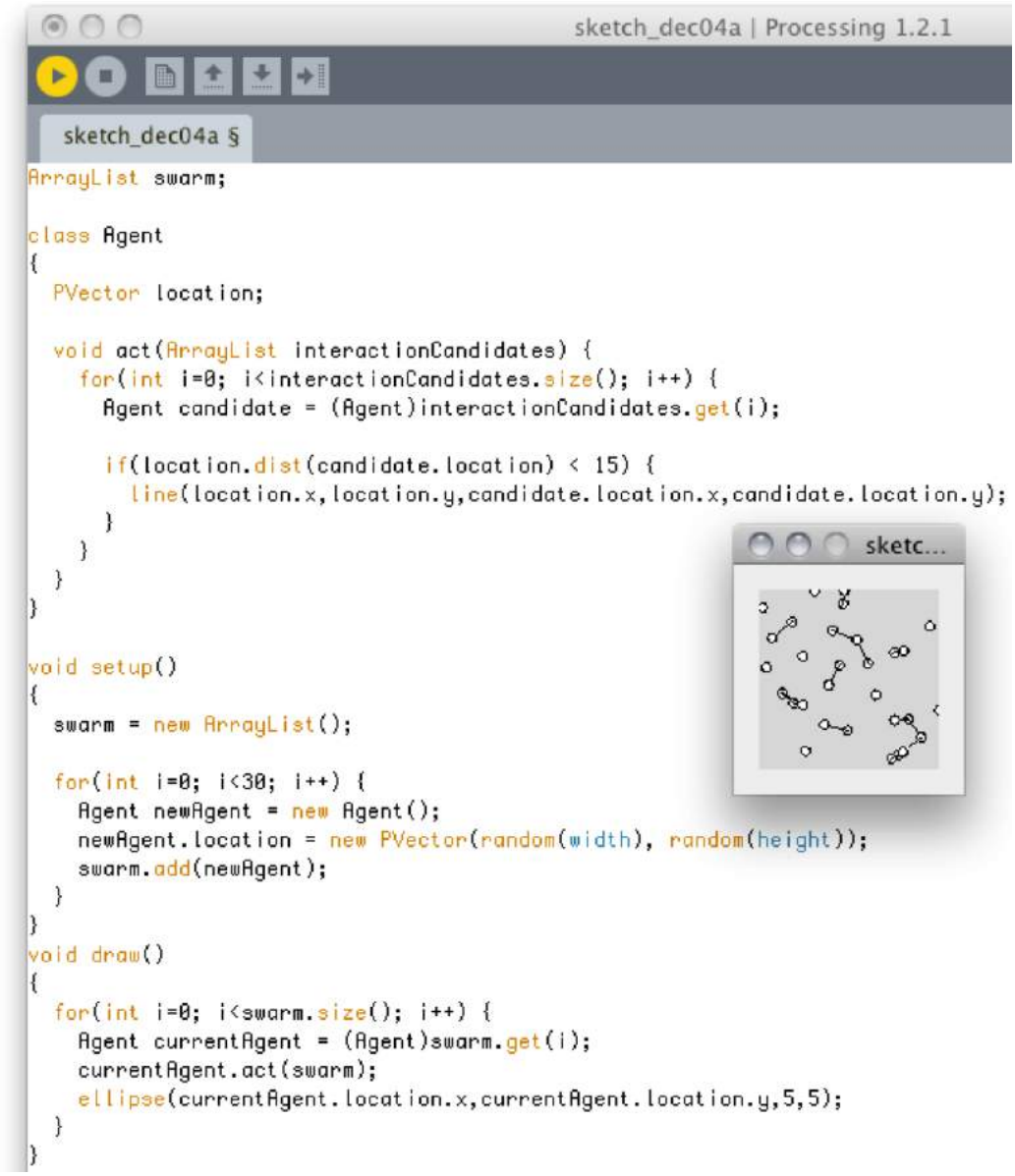
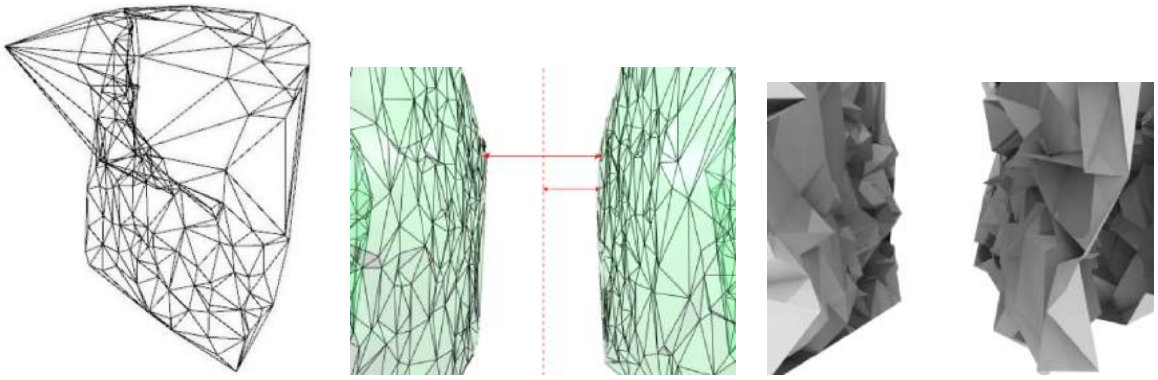


Figure 6.2: The complete Processing code of a basic swarm-programming infrastructure. The draw() method executes the act() method of a list of Agent objects.

6.4.1 Sentient Surfaces

Julie Brache and Michael Scantland worked on an extension of the previous programming example. Agents serve as the vertices of a mesh and their neighbor relations translate into the mesh topology. Movements of agents can thus dynamically reconfigure the surface. Fig. 6.3 shows a basic setup of an according simulation of sentient surfaces. Fig. 6.4 depicts exploration states of the emerging mesh dynamics.

Scantland and Braches 2010 studio project paralleled the sentient surface investigation through



(a) (b) (c)

Figure 6.3: In sentient surfaces, agents serve as mesh vertices and their movements reconfigure the structures. (a) A single agent drags its neighbors out of the mesh. (b) Conceptual illustration of perception thresholds between two sentient surfaces. (c) Attracting and repelling forces among the agents result in rough surface configurations.

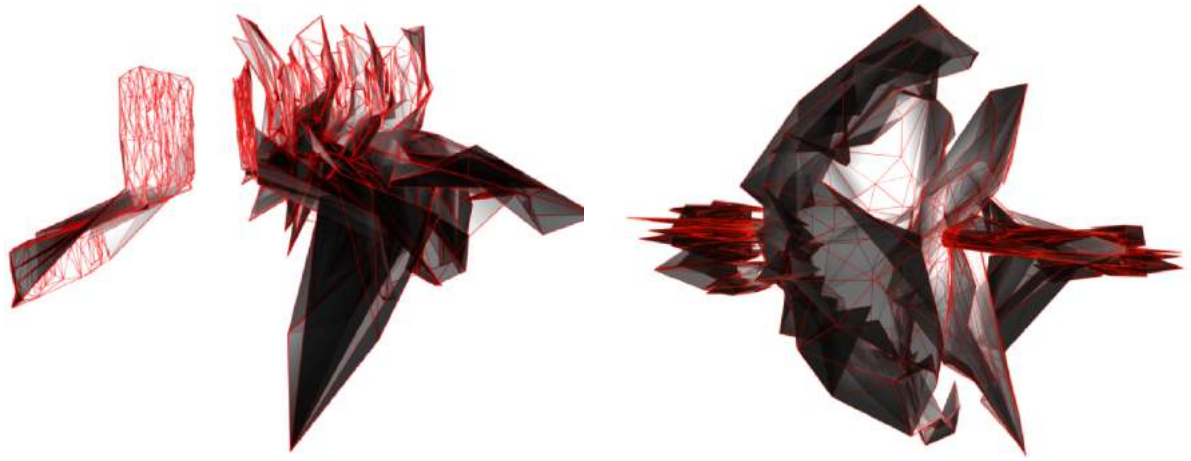
the deployment of a nodal network distributed throughout the tower site. While addressing a different frequency and scale of modulation in the studio project, displacement of interior spaces and replacement of exterior structure formed an integrated relationship between differential programs. Fig. 6.5(a) diagrams the responsive relationship between exterior structure, interior space and building envelope generated by grafting the two systems together. Fig. 6.5(b) illustrates the exterior view of the tower as the nodal network weaves through the building.

6.4.2 Creating Space through Diversity

Ryan Palibroda approached the organization of land occupation from a 2D perspective. Agents keep pushing each other in accordance with their preferences until a steady state is reached (Fig. 6.6).

Ryan Trefz also experimented with different agent types (Fig. 6.7). In addition to repelling forces, Trefz relied on the whole array of boid urges to inform his agents flight: alignment, cohesion, separation [399]. Differently configured agents can be distinguished through size and hue.

Trefz and Palibroda collaborated in studio to produce a tower whereby the building exterior operated like a solar landscape. While the exterior borrowed tactics from Palibrodas displaced fields (Fig. 6.6(a)), interior spaces were formed by tracing flocking positions into structural networks (Fig. 6.8(b)).



(a) (b)

Figure 6.4: In sentient surfaces, agents serve as mesh vertices and their movements reconfigure the structures. (a) A single agent drags its neighbors out of the mesh. (b) Conceptual illustration of perception thresholds between two sentient surfaces. (c) Attracting and repelling forces among the agents result in rough surface configurations.

6.4.3 Carving Structures

Chris Vander Hoek explored subtractive generation of architectural spaces. In his project, commuting swarms (Fig. 6.10(a)) carve out cubic volumes that recursively decompose into eight smaller cubes on collision with a swarm individual (Fig. 6.9).

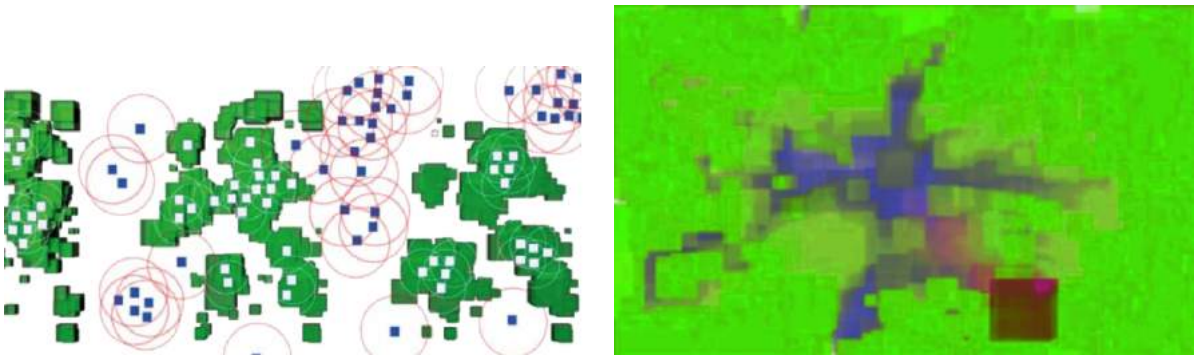
In his studio project, he used the same particle swarm to generate a 3D voronoi extrusion from the building base in order to extend the interiority of the tower into the adjacent plaza (Fig. 10(a)). Physical explorations focused on fabrication and assembly techniques that subtract from adjacent spaces.

6.4.4 Procreating Particles

Jonathan Choo and Fadilah Hamid applied their knowledge of agent-based modeling in a simulation written in MEL, the embedded scripting language of the Maya rendering software. A predefined space is populated with agents (Fig. 6.11(a)) that attract and repel each other (Fig. 6.11(b)) and procreate on collision. The inter-agent relations translate into a smooth surface with hollow spaces (Fig. 6.11(c) and Fig. 6.12).

(a) (b)

Figure 6.5: (a) Grafting Strategy (b) Exterior Perspective.



(a) (b)

Figure 6.6: (a) Agents of a specific type form clusters as they push agents of other types away. (b) Opposing forces between different agent types result in organically shaped high-density areas.

6.5 Explorations of Biomimetic Design: Iteration 2 (2012)

A second set of architectural research studio explorations (Integrative Intelligence) took place in 2012 that continued the original line of biomimetic investigations. More precisely, the studio sought to find new application for agent-based processes within the historic Pruitt-Igoe site which were submitted (mid-term) to the Pruitt-Igoe Now competition (`http://www.pruittigoenow.org/`). The selection of work included in this section demonstrates the continuing developments of the biomimetic investigations initiated between the authors, particularly as they relate to material experimentation that were not tackled in 2010s architectural research studio.

(a) (b)

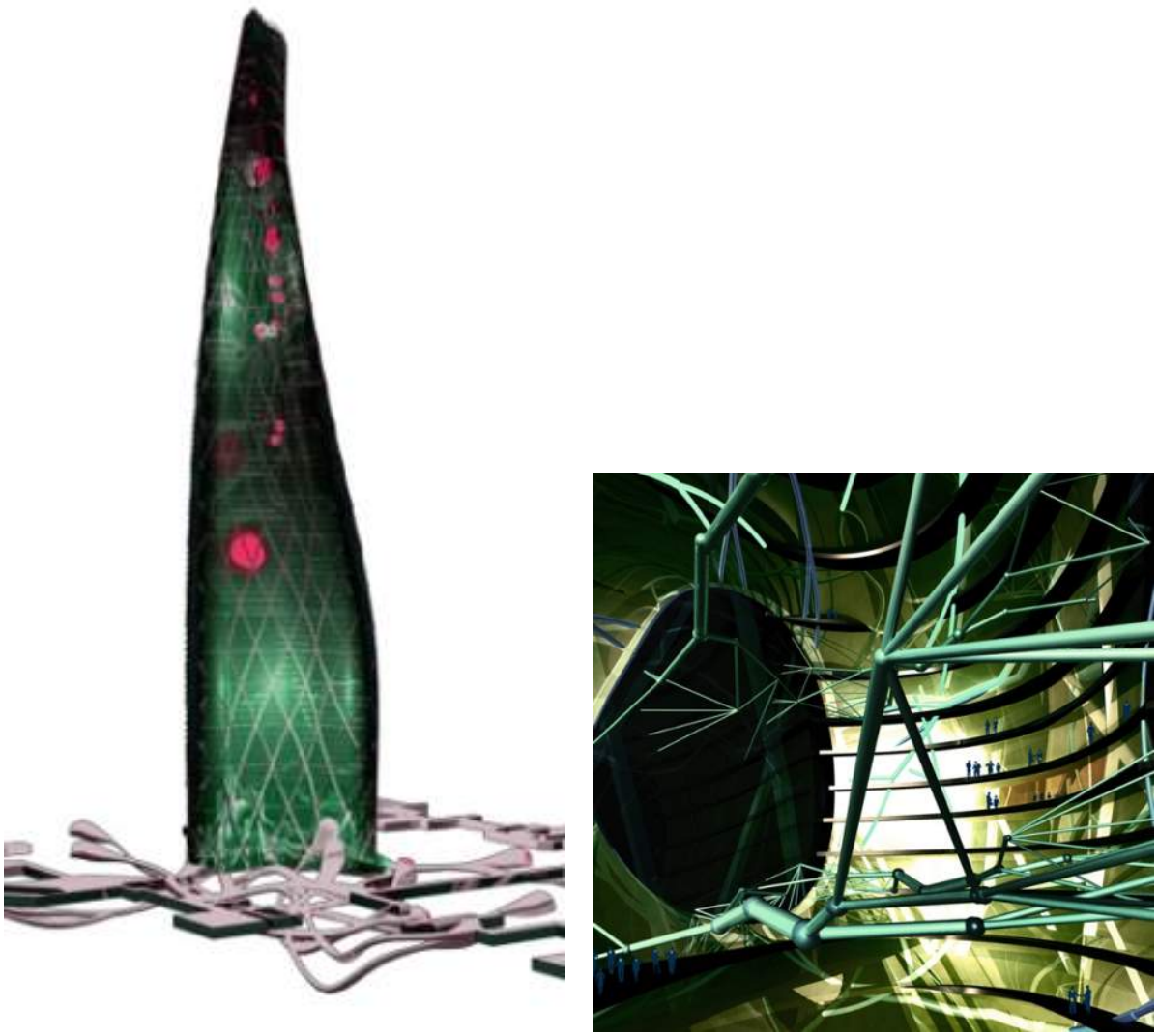
Figure 6.7: Cluttering and clustering flocking formations to inform a dynamic architecture inspired by Craig Reynolds boids (1987) and Nicholas Reeves Mascarillons (2005).

6.5.1 Decay Swarms

Jodi James advances the simulation and projection of sentient surfaces into her 2012 studio project whereby growth and decay are mobilized against one another across building surfaces both internal and external (Fig. 6.13). Using a combination of swarming and grafting definitions previously developed by Taron, James reprogrammed the material, structural and programmatic organization of a large-scaled historical housing project, transforming it into a self-contained, self-regulating society (Figs. 6.14-6.16). Through material experiments, James was able to combine CNC milled high density foam with the precise application of acetone to produce the kind of surface effects produced through her computational simulations (Fig. 6.17). These tests when taken further, were able to articulate large territories of decayed built space (Fig. 6.18). James project allows for innovative new ways to not only make new buildings but also new methods for modifying existing structures. This is becoming an increasingly important territory for research as the reuse of existing buildings offers distinct advantages when it comes to sustainability and life-cycle assessment.

6.5.2 Spherical Aggregations

Erin Faulkner de Gordillos project focuses on particle simulations that produce forms similar to Choo and Hamids Procreating Particles (Figs. 6.11 and 6.12) but goes further to emphasize the possibility for their material mass production. The aesthetic similarities to Choo and Hamid are most evident in the general project imaging (Fig. 6.19).

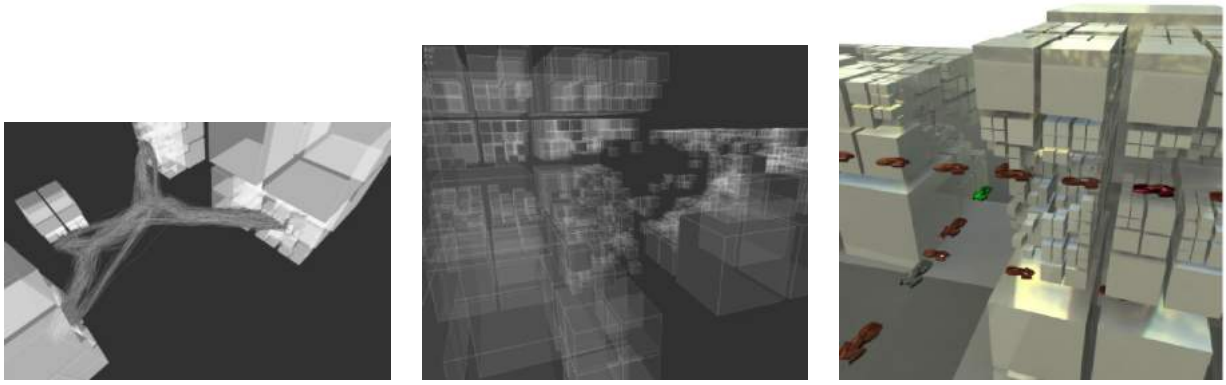


(a) (b)

Figure 6.8: (a) Hotspots embedded within the building facade operate as attractors for (b) interior conditions that trace the position of flocking particles through space.

However, investigations moved very quickly toward ways in which spherical particles could aggregate and intersect materially such that the spatial geometry could be achieved. Toward this end, de Gordillo gravitated toward flat stock and laser cutting in order to run tests. These tests were initiated using a 2-dimensional logic that would embed 3-dimensional connection strategies in order to form spatial conditions (Fig. 6.20). This project lends itself particularly well to stress the transition from local, micro-interactions toward macro-assemblies.

Tests were procedurally tracked as the individual assembly grew and aggregated (Figs. 21 and 22). The complex, emergent form tasks the designer with extensive explorations of generated perspectives. De Gordillos work, while interesting in its own right, is perhaps most interesting in the fact that the parts themselves contain the logic for assembly without specifying exactly



(a) (b) (c)

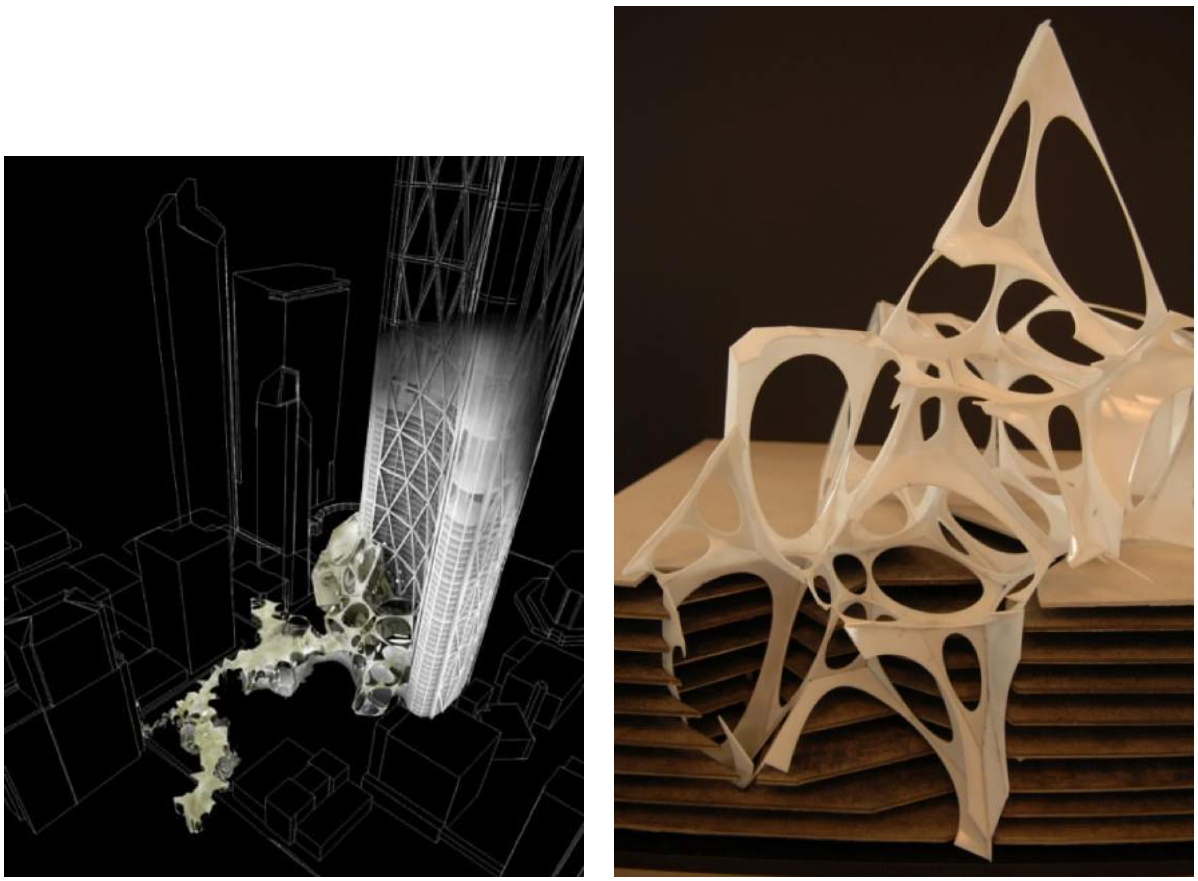
Figure 6.9: (a,b) Commuting swarms carve out cubic volumes. Upon collision between a swarm individual and a volume, it recursively decomposes into eight cubes until it completely disappears. (c) Future city optimized for mid-air traffic flow.

what form may come. This may provide the best insight into how future agent-based systems will be assembled: absent a dimensioned building envelope, but rather a series of parts that can be organized and reorganized to yield [re]configurable spaces, programs and form.

6.6 Program Evaluation and Future Work

After two iterations of our trans-disciplinary program, we have been able to identify and correct some of its weaknesses and exploit some its strengths. Success manifested through the combination of algorithmic and biological foundations offered through the Computer Science course while the Architecture research studio provided a space for exploration and application of those techniques in the context of the built environment. Problems developed in the studio were in turn framed as means for evolving the projects in the Computer Science course. We found that enthusiasm was renewed on both fronts with the constant unfolding of new problems, innovative solutions and range of applications.

In the first iteration of the program, the Computer Science projects did suffer from starting only after a number of weeks of introductory coding exercises were completed. As such, the students projects did not have the full term to evolve and develop. We amended this issue in the second iteration by employing instructor-generated podcasts of basic lessons (`http://www.youtube.com/user/svonmammen`). At the same time, coding basics were conceptually taught in class; however, programming training was provided on-demand in addition to the



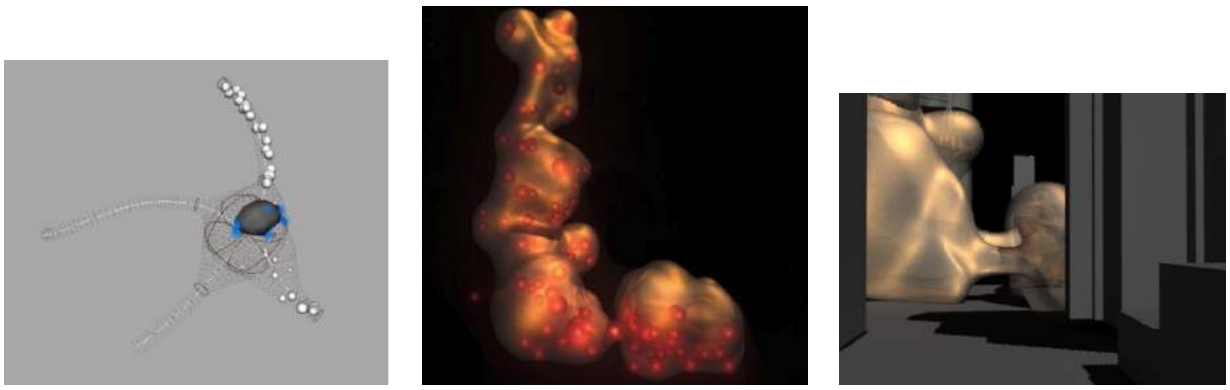
(a) (b)

Figure 6.10: (a) Aerial perspective (b) Sectional study model carves away from the subterranean parking lot below the site.

lectures in order to free up time and let the student projects start early in the term. Obliging each student to pursue his own project (as opposed to group-projects of two) turned out to be another major improvement in terms of personal motivation and creativity.

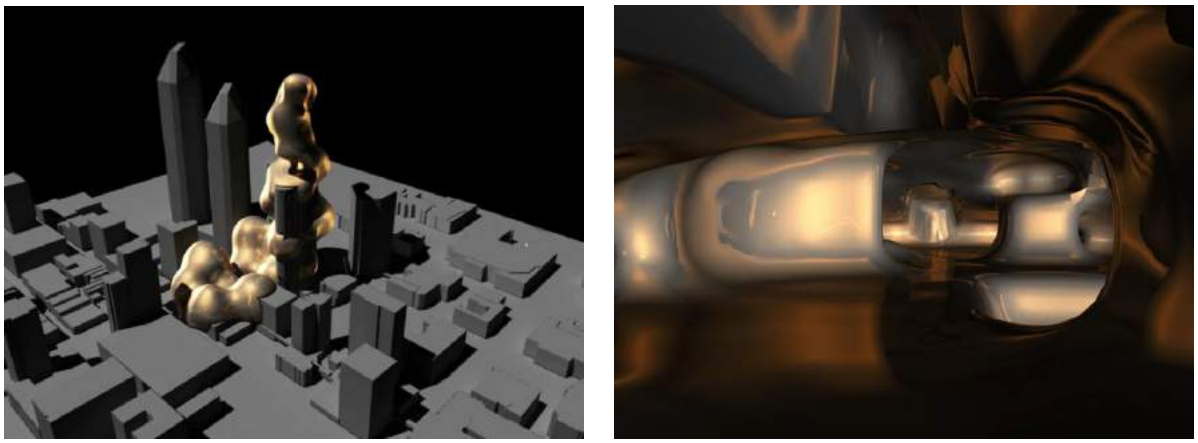
The Architecture research studio did provide an appropriate moment in the curriculum to engage in computational biomimicry (in the last term of a six-term graduate program) given a developed knowledge during their first two and a half years of graduate education. However, the Computer Science course is seen as introductory and would certainly serve as a valuable skill to have earlier in a graduate curriculum (Architecture, Computer Science or otherwise). A looser but perhaps more profound connection might exist between the Computer Science course and an Architecture studio positioned earlier in a graduate program, thus giving students more opportunities to experiment with and develop their coding skills as they continue through school.

We view the premise of decentralized control in both Computer Science and Architecture as



(a) (b) (c)

Figure 6.11: (a) Slowly a predefined volume is populated with agents (represented as spheres). (b) Attracting agents are colored in bright red. (c) A smooth mesh encloses the interacting agents.



(a) (b)

Figure 6.12: (a) An architectural site is redefined by interacting particles. (b) The interior space of the resulting space.

fundamental to the advancement of our own research and in both disciplines at large. By producing a pedagogical framework whereby swarms, natural systems and Architecture operate within an interchangeable space, each can inform the others in unique and useful ways; envisioning biomimetic code as Architecture, Architecture as nature, and nature as codified milieu. While the courses reinforce one another by structuring the exchange of information between one another, less resolved are the structures that might produce continuity and evolution from one year to the next. By archiving code packages developed in previous course iterations, incoming students have shown to carry on and develop those definitions further, hybridize multiple definitions together or if nothing is attractive to them, start something from scratch, thus broadening the gene pool. The code developed in a given term has the chance to go on living after a course iteration has ended which stands metaphorically for our approach

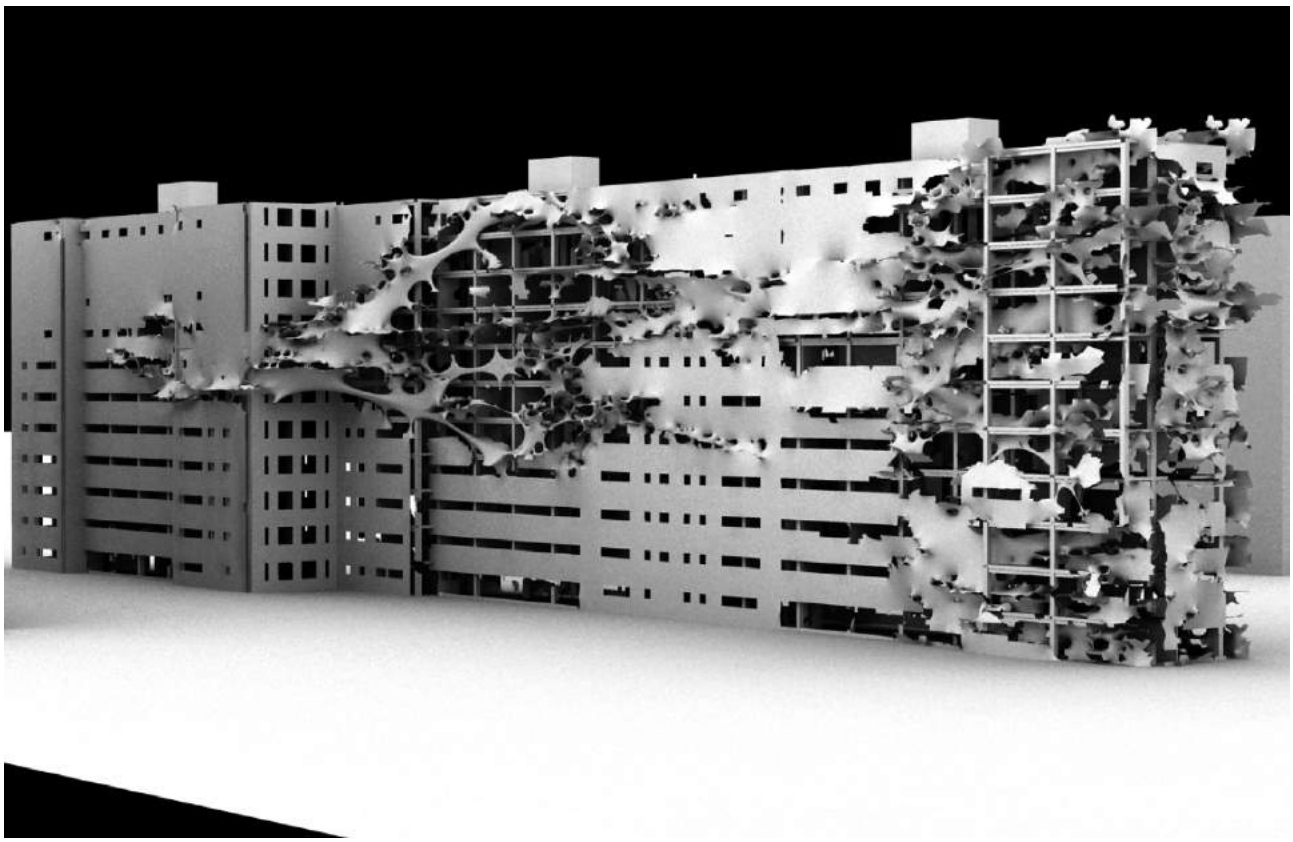


Figure 6.13: Swarm-generated growth and decay model.

to trans-disciplinary teaching and research.

Another line of continuation might make a selection of previously developed definitions available to an Architecture research studio with the charge that they design with/make use of them in an architectural capacity. This is already a model in use whereby swarm code developed by Taron is released to Architecture students for use as a generative design tool. Interoperability between agent-based models and analytical software could also prove useful in fostering emergent performative capabilities of these models whereby swarms would generate fitness values through evolutionary feedback loops. von Mammen has been part of another trans-disciplinary project for the past three years to develop according modelling languages and computational frameworks (`http://lindsayvirtualhuman.org/`). This looks to be a promising trajectory as analytical tools such as finite element analysis, computational fluid dynamics and energy performance software have come into the mainstream of architectural design process.

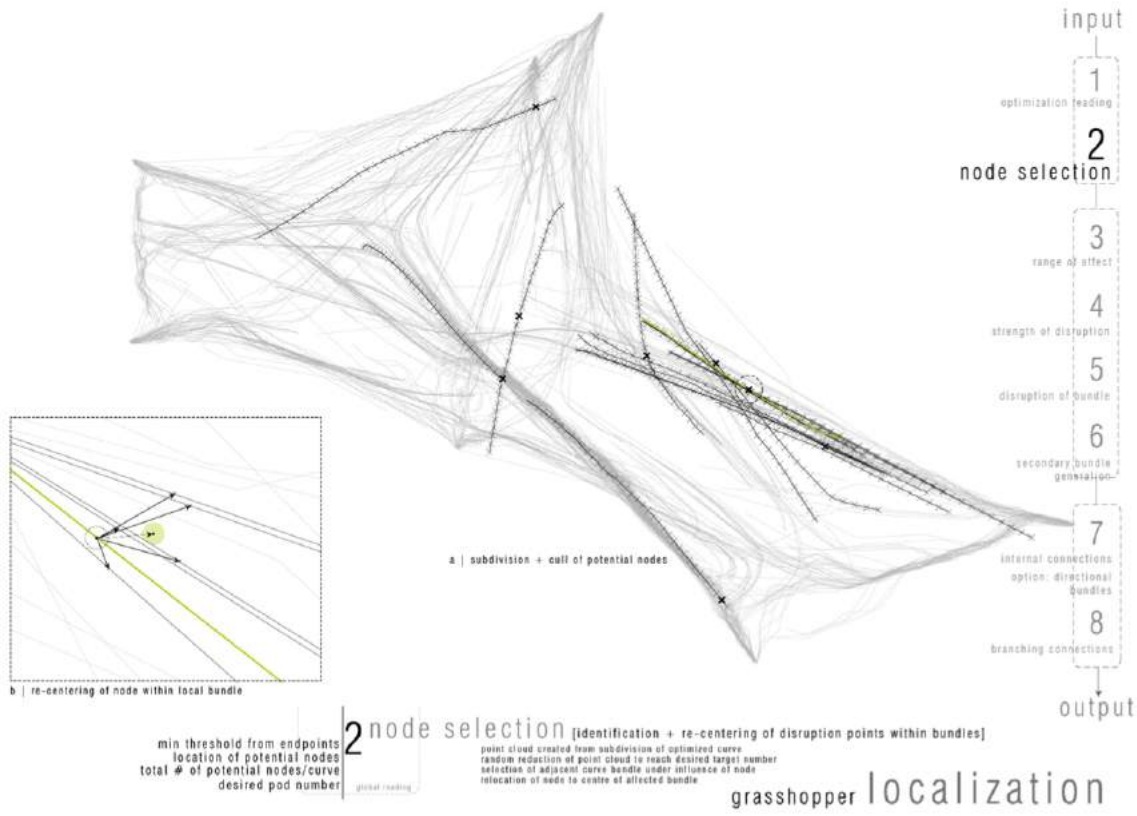


Figure 6.14: Parametric Grafting Diagram (internal).

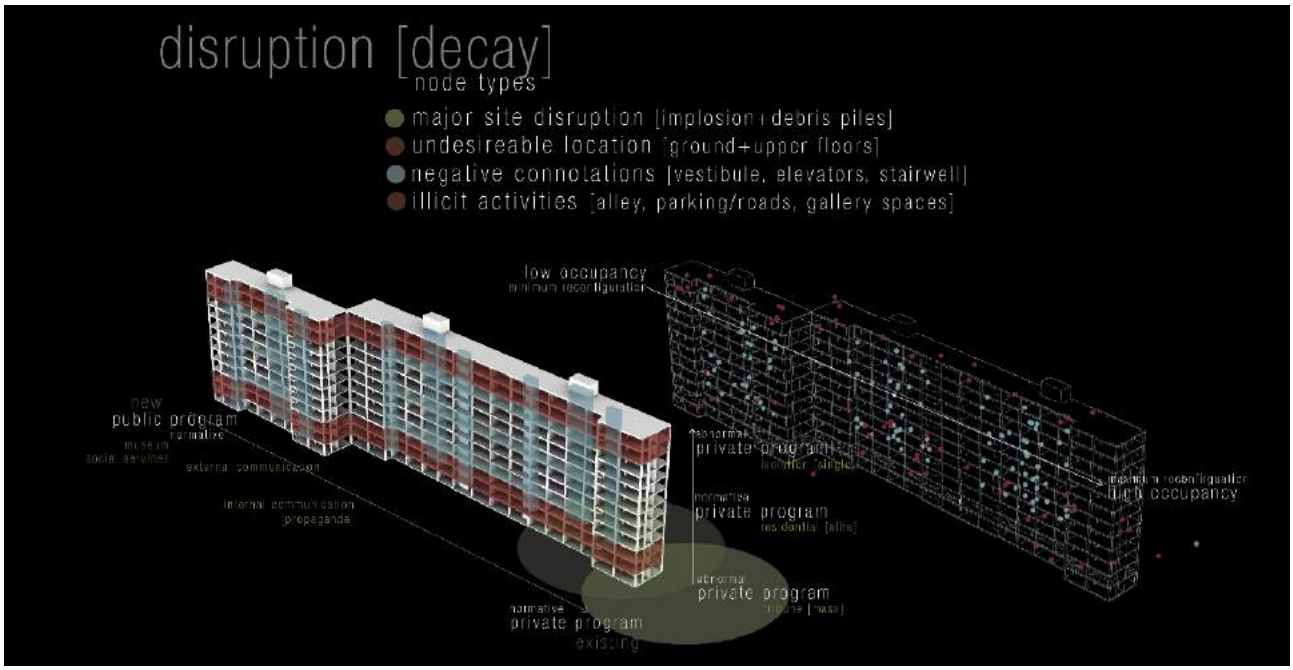


Figure 6.15: Axonometric Program Diagram.

Figure 6.16: Sectional Program Diagram.

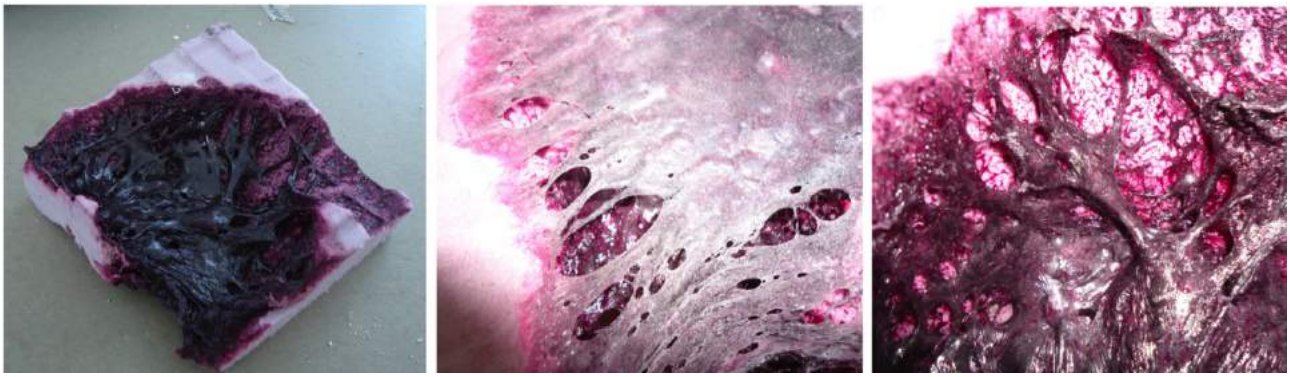


Figure 6.17: Dense Foam Decay Tests.

Figure 6.18: Project Model.

Figure 6.19: Project Render.

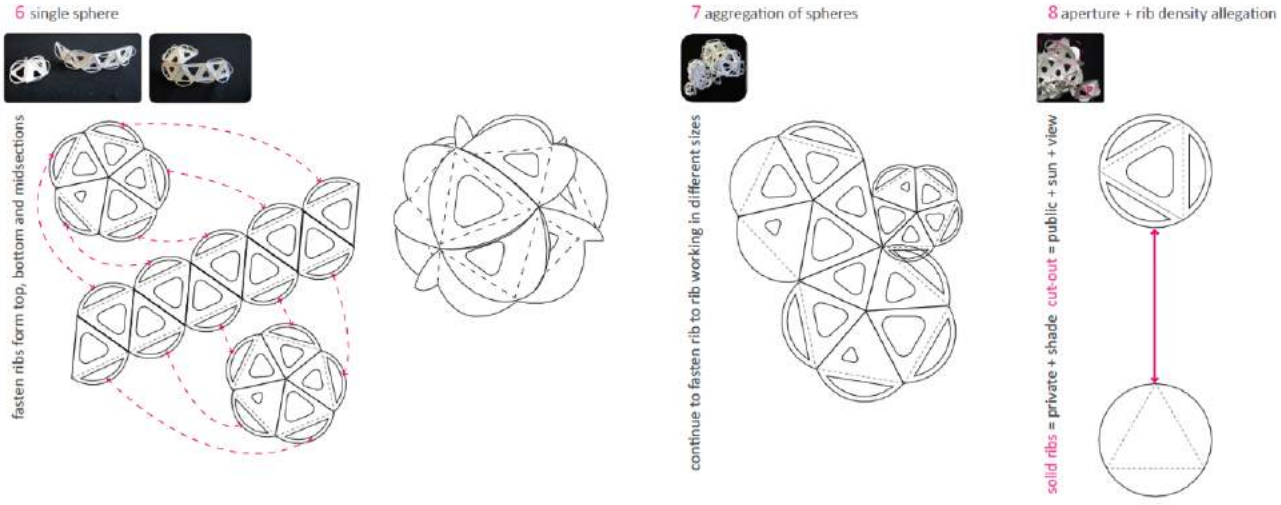


Figure 6.20: Part-to-whole Assembly Diagram.

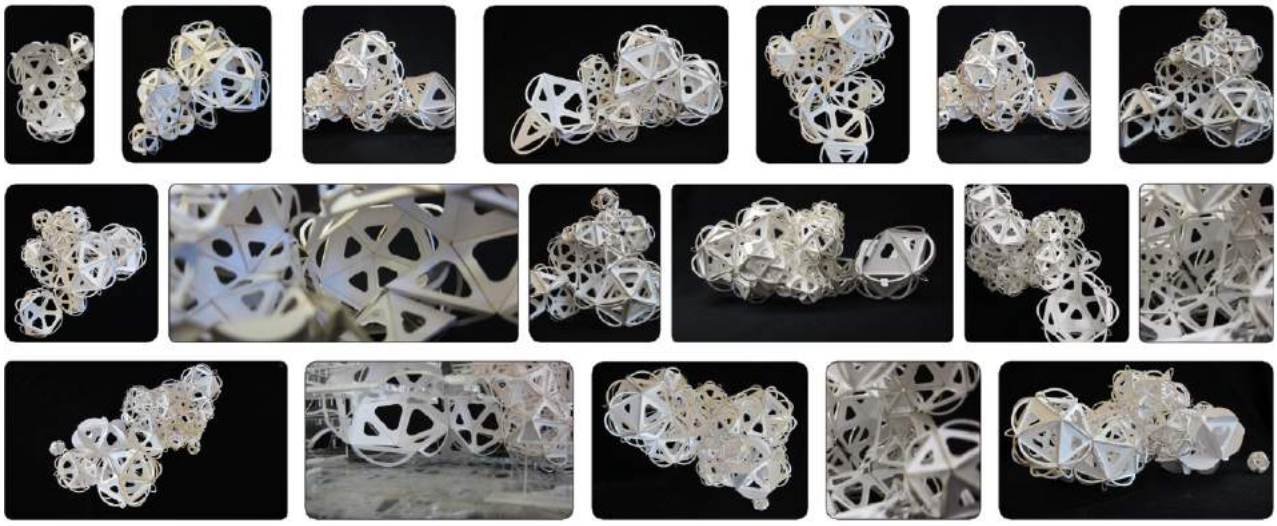


Figure 6.21: Aggregate Assembly Sequence.

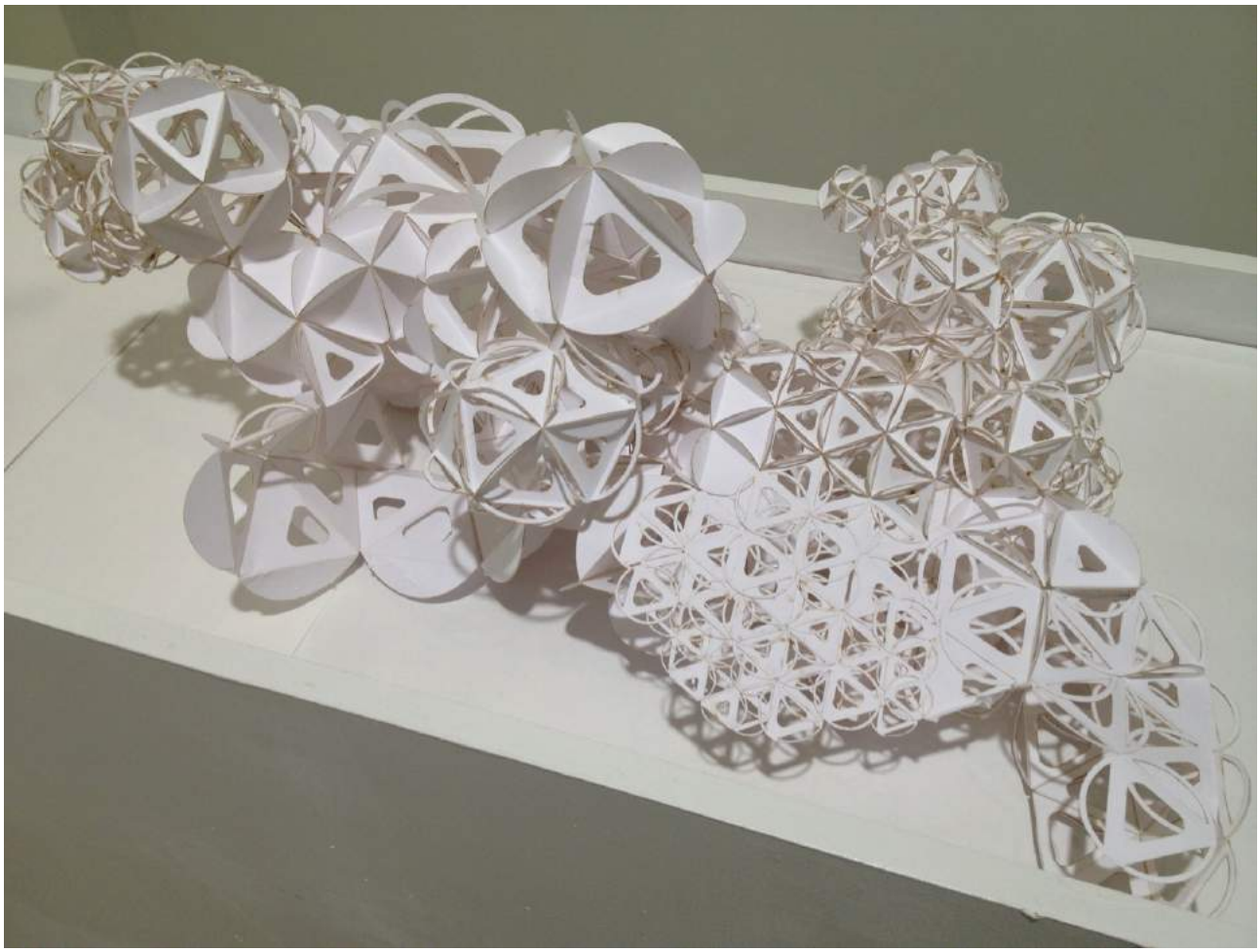


Figure 6.22: Final Model Image.

Chapter 7

Artistic Exploration of the Worlds of Digital Developmental Swarms

We present art work that was inspired by a computational model called Swarm Grammars. Herein, the ‘ liveliness’ of swarms is combined with the generative capabilities of more established developmental representations. Three artists followed their very individual approaches to explore the creativity and dynamics of Swarm Grammar structures. One artist chose to interactively breed structures to compose virtual spaces. Work by the second artist explores the movement and construction dynamics of the interactive swarms. The third artist translated developmental processes of Swarm Grammars into interactions of paint particles driven by friction and gravity. Swarm structures transcend from virtuality to sculptural manifestations.

Sebastian von Mammen, Joyce Wong, Thomas Wissmeier, Christian Jacob: *Artistic Exploration of the Worlds of Digital Developmental Swarms*, LEONARDO 44 (2011), 513.

7.1 Introduction

With an open mind and open eyes, we discover breathtaking forms and colors, textures and shapes in nature. The ridged desert sands, the reflection of the sun in the sea, or serene alpine landscapes have served as motifs for countless paintings, pictures and movies. Nature’s glamorous constituents of organic origin, like butterflies and shells, have been collected for their beauty over centuries. Through algorithms that retrace the processes of growth of plants and

other forms of life, digital art has grown far beyond a binary heritage that only suits re-shaping the reflections of real objects. Nature-inspired processes of growth and development may be algorithmically implemented simply by repeated substitution of symbols in a string according to a set of rules. L-Systems, which follow exactly this idea, have been applied to simulate the growth of cells [400], plants [338], organs [401], and architectural designs [402]. From a larger perspective, generative processes or developmental models have extended the computational realms of creation and creativity [403].

Besides the distinction of individual symbols, e.g. A or B, any information about the relations among the replicating units have to be inferred from the L-Systems' strings by an external interpreter. In Swarm Grammars (SGs) complex networks of relationships develop because SGs combine the developmental aspect of L-Systems with an agent-based modeling approach [399]. Each symbol is considered an individual that perceives and reacts to stimuli in its environment while grammatical rules drive its reproduction and construction processes. Thus, highly dynamic, complex networks of interactions emerge [117].

The SG examples in this article combine the coordination of movement as seen in birds, the reproduction as modeled in the growth of cell colonies or plants and the indirect communication through the environment investigated in social insects. These aspects are found in many natural systems, e.g. chemotaxis, quorum sensing and bio-film aggregation at the cellular level. Algorithmically, those aspects have mainly been studied independently. We utilize Swarm Grammars that combine these aspects to create life-like artefacts that resemble organic forms [343] that capture the dynamics of the construction processes [347].

While artists and computer scientists have explored computational developmental models and artificial swarms to a large extent (Section 7.2), the exploration of Swarm Grammars has begun only very recently. Nest constructions by social insects like wasps, ants and termites, however, already hint at the power of swarm-based developmental models. Details of the inspiring biological models are provided in Section 7.3. Subsequently, works by three artists are presented, whose artistic endeavors might be summarized as (1) utilizing Swarm Grammars to create artificial spaces (Section 7.4), as (2) inspirational manifestations of interaction dynamics in complex systems (Section 7.5), and as (3) visionary playgrounds for artistic expression, media and forms (Section 7.6). Finally, in Section 7.7, we briefly recapitulate the highlighted aspects of the presented art in the context of computational developmental models and provide an

outlook on possible future work.

7.2 Related Work

Machines help us in performing repetitive tasks. The more powerful the machine, the more complex the conductible action can be. Analogously, the more powerful a computational representation, regarding its expressiveness and abstraction, the more complex the computable result will be. Multi-cellular organisms are outstandingly complex systems. With L-Systems, Lindenmayer successfully found a grammatical representation for the growth of clusters of cells [392]. In retracing the development in natural systems, L-Systems can create natural aesthetics. Based on L-Systems, and other mathematical methods—including fractals—intriguing sculptures of artificial aesthetics can be evolved [404]. Whole worlds of seemingly living organisms can be made to grow in virtual spaces [405].

An important aspect of developmental systems is their ability to be evolve. Real creativity is conjured when computational evolution drives the development of structural and graphical representations. The necessary selection processes of fit individuals can be performed manually, by a human 'breeder' [406, 407]. Alternatively, automatic evolutionary runs can foster structural attributes that optimize mathematical fitness measures, e.g. for increasing the design complexity [369] or to introduce architectural functionality. In an attempt to automatize the evaluation of generated computer art, human-made works can provide reference points for assessments. A lot of artistic works have been produced through the combination of computational evolution and developmental models, see for example Part III in [408]. However, artificial life methodologies have instilled other important aspects in computer arts. Artificial swarms, for instance, regularly enthuse public audiences in interactive art installations [409]. Through the interactions of large numbers of flocking individuals, or boids (see Reynolds [399]), a certain ' liveliness' is communicated. As an underlying principle, each swarm individual (referring to a single unit of the swarm) has a limited perceptional field that determines its neighborhood. In Figure 7.1 the field of perception is defined by the viewing distance d and the angle α. Furthermore, at each time-step of the simulation, the individual computes its current acceleration in accordance with its neighbors. As a consequence, the individual changes its position and its neighborhood configuration, as do the neighboring flock mates. Hence, a feedback loop of

interactions is triggered that can lead to a variety of flocking behaviors [376].

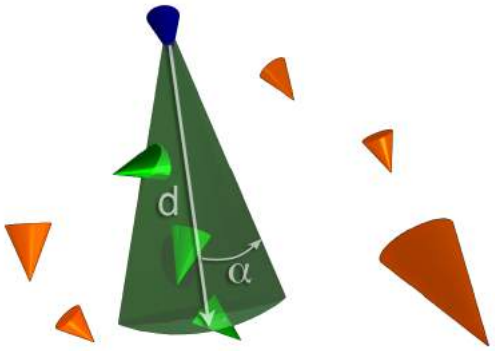


Figure 7.1: All mates within the conic field of perception of the dark agent are considered its neighbors. The perception range is determined by a distance d and angle α.

Primarily, the flocking dynamics of artificial swarms feed into animations. But the spatio-temporal interaction network of swarms also supports computational music generation [374]. Furthermore, simply by leaving traces in space, the flocking patterns solidify and their dynamics can be captured in three-dimensional sculptures. Some of these virtual sculptures have served as inspirations for traditionally crafted collages and paintings, see von Mammen [347].

7.3 Developmental Creativity of Swarms

When termites build their nests, they do not work from a blueprint of the new construction. Instead, they follow their instinct and build wherever they see fit. In fact, the construction activities of social insects are determined by pheromone trails left by other individuals, construction cues in their immediate environment, physical gradients such as heat and humidity, or by plain chance. Based on this idea, simple sets of probabilistic behavioral rules have led to constructions similar to those observed with ants and wasps [230].

Swarm Grammars are a computational concept that integrates (1) the power of constructive swarms with (2) the ability to instantaneously reproduce and (3) the boid flocking paradigm outlined earlier. Figure 7.2 shows an example of Swarm Grammar (SG) development. Through Figure 7.2(a) to Figure 7.2(g) SG agents split and leave construction elements along its paths. With an initial, " axiomatic" agent, A, reproduction is performed according to the rule-set: $\{A \rightarrow ABC, B \rightarrow A, C \rightarrow AA\}$. In Figure 7.2(a), a construction element has been placed by

an agent of type A. Already, the first production rule has been executed, leaving three agents A,B,C, illustrated as the small pyramids, floating on top of the construction. Subsequently, in Figure 7.2(b) to Figure 7.2(g) the reproduction and construction process continues: Agents B and C quickly drift apart, loosing most of their internal energy. The loss of energy is reflected in the shrinking diameter of their constructions. Hence, after another reproduction on each side, the construction processes stop. Only the initial agent A keeps enough energy for persisting cycles of reproduction and construction. In the used SG configuration, the loss of energy is type-dependent and linked to the agents' movements and replication processes. The rules for reproduction and differentiation are triggered after a type-dependent interval of simulated steps.

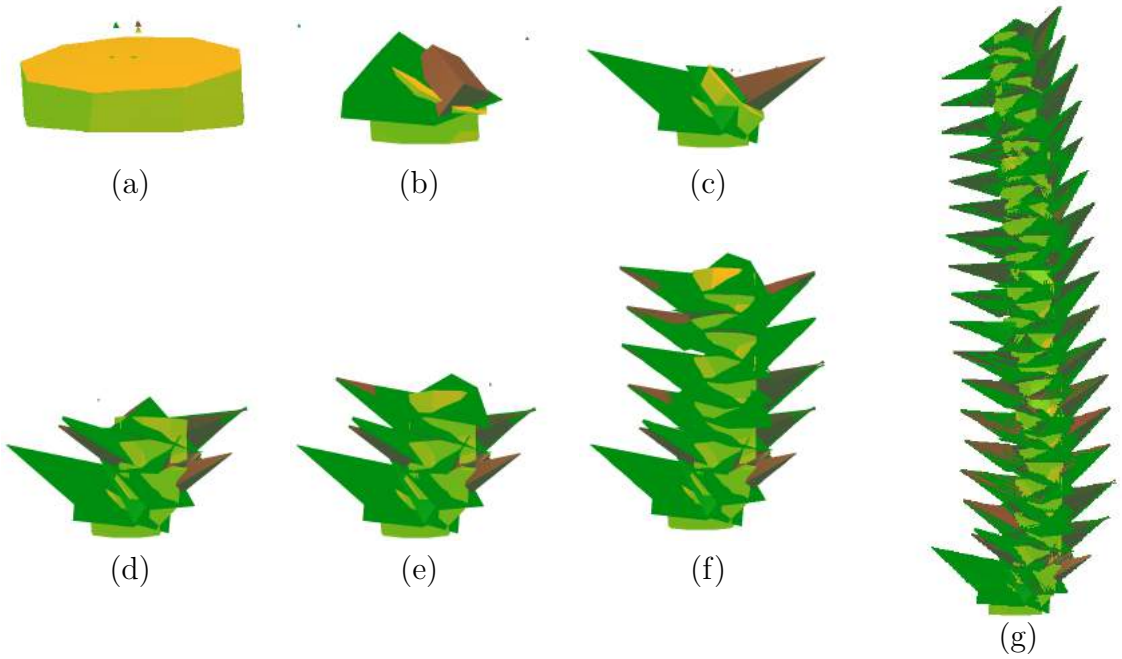
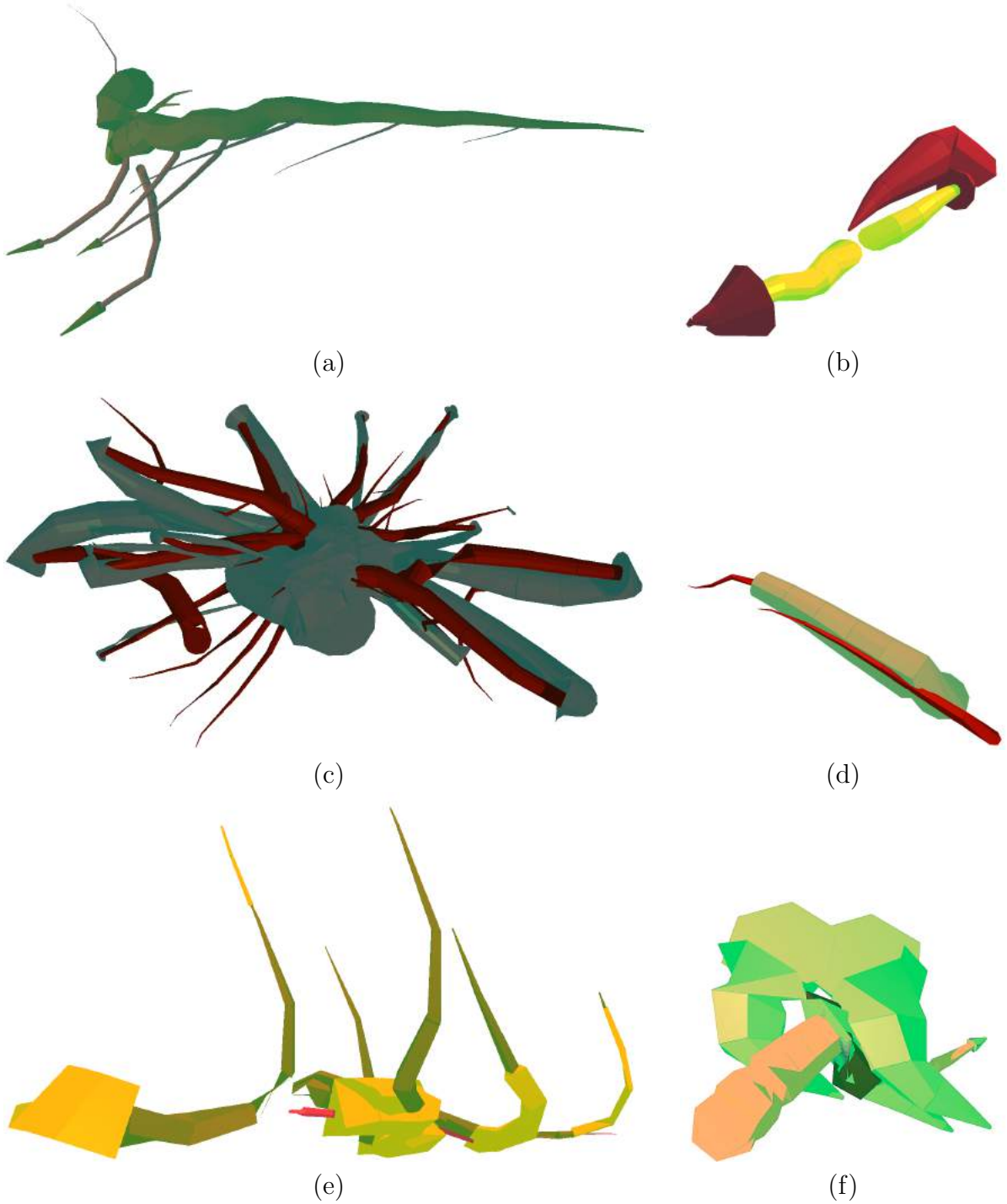


Figure 7.2: Swarm Grammars in Action: (a) Three agents (represented as small pyramids) are heading upwards after their initialization. (b) The green and brown agents drift apart. Before they run out of energy they create the spiky construction elements seen in (c). The spikes occur because the agents' energy levels are linked to their construction elements' diameters. In the meantime, the yellow agent has produced new offspring and the construction module created so far is repeatedly added to the growing structure (d-g).

Different sculptures emerge, when we change the reproduction rules and the agent properties (Figure 7.3). This SG configuration can be changed manually, but also by means of computational evolution. Many approaches have been tested for the latter case: (1) Through interactive evolution an external breeder rates an array of SGs (see [343]). (2) In an immersive (co-)evolution approach a gardener tinkers with SGs in a virtual space by replenishing their

energy to further their growth, by inducing mutations, or by crossbreeding selected specimen [344]. (3) Using automatic evolution a fitness measure is formulated mathematically that serves to evaluate the construction processes and the emerging structure [345].



(a) (b) (c) (d) (e) (f)

Figure 7.3: Evolved Swarm Grammar Samples.

7.4 Creating Spaces

Stimulated by the architectural capabilities of Swarm Grammars [345], the artist (S.v.M.) combined swarm structures to create surreal, artificial worlds. In about 40 interactive evolutionary experiments, the artist bred the utilized Swarm Grammar structures, relying on the *Mathematica* library *Evolvica* for the evolutionary algorithm [365] and the user interface *Inspirica* [376]. The breeding experiments yielded the sets of Swarm Grammar structures displayed in Figure 7.4(a) and (c). In the corresponding paintings, Figure 7.4(b) and (d), a chameleon and a bighorn sheep are immersed in complementary artificial environments. According to the order of appearance, all figures in this article show the computer-generated contents first, followed by their human-made artistic realizations.

During the evolutionary runs, the artist followed two main objectives. Firstly, robust looking beams should emerge that form a structural mesh, thus opening vast spaces by their mere existence. Secondly, fuzziness, continuity and the resemblance to organic forms should warrant the authenticity of the generated virtual worlds. The color gradients in the backgrounds reflect the extreme climates of the habitats of the projected animals. They also highlight the sound, wholesome, fluent structural architecture in Figure 7.4(b) and the liveliness and dynamics caught in the erratic structures of Figure 7.4(d) with “warm” and “cold” palettes, respectively.

7.5 Abstraction of Dynamics

The smooth, interwoven curves with sporadically grown thorns seen in Figure 5(a) and (b) emphasize the emergent flocking dynamics of the constructing Swarm Grammar. Figure 7.5(a) shows a screenshot from the interactive breeding procedure: Several specimens are growing according to their configuration, independently in isolated spaces. After close inspection, the external breeder rates the presented Swarm Grammars. Based on the received fitness the individuals are selected for the next generation of Swarm Grammar simulations. During the transition from one generation to the next, a certain percentage of selected Swarm Grammars are interbred, to combine some of their characteristics. The so-called mutation operator introduces small configuration changes to some of the other selected specimens.

The artist (J.W.) chose the SG structure depicted in Figure 7.5(b) to inspire the real-world sculpture shown in Figure 7.5(c) and (d). Four glass plates, painted with matching SG graphics, are connected through bearings that allow to turn them (Figure 7.5(d)).

Through the innovative sculptural design, the dynamics of the Swarm Grammar growth processes are maintained. The interdependent flocking of the SG agents can be replayed by rotating the glass plates and seeing a great number of alternative structures of the same constituents emerge. The layer-wise mapping of a two-dimensional image resulted in a fully three-dimensional sculpture, as both the glass plates and the bearings have a significant height (about 0.3 inches). Hence, although the images are flat as on the computer screen, they materialize as three-dimensional due to the interplay of transparent plates and bearings.

7.6 Models as Basis for Experiments

Publishing about art inherently requires photographs to show the discussed pieces. In the following paragraphs, however, the artist (T.W.) intentionally chose the medium of photography for a reason. Especially in works that are based on sculptures, presented at the end of this section, photography supports the artist's experimental work with an immense degree of freedom. As pointed out by the Austrian arts professor Christian Reder [410]:

> As a multitude of considerations, pictures, layers, associations, the model is always a working model at the same time.

This statement is an imperative—to use the model to work with it and to improve on it, instead of chasing its immediate finalization. In the context of the presented work, a cascade of modeling indirections has hurried ahead: Biological models that were unified in a computational (meta-)model of developmental processes. In a second step, unfolding an instance in the computational model space led to three-dimensional structures in virtual space that served as models for artistic work. Now, the importance of Reder's statement lies in the freedom of the artist: No formal stipulations cross the artistic path. But, if so chosen, insight can be pursued with intrinsically motivated playfulness regarding perspective and perception. This deems to be even more important when striving for comprehension of interwoven complex models.

7.6.1 Translating between Virtual Spaces

The Swarm Grammar sculpture of Figure 7.6(a) is captured in a rather abstract fashion in the photograph shown in Figure 7.6(b): With monochromatic coloring and extrapolation beyond the original structure by adding a third column. Like the art pieces presented in Section 7.4, the photograph accentuates the architectural characteristics of some Swarm Grammar structures. But even though the stylized elements were reduced in number, the architectural, sculptural features of the original structure are furthered. The underlying material piece was painted with varnish on a large wooden panel. It possessed strong contrasts and hard edges that were smoothened in order to facilitate the engagement in the photograph's spatial aspects.

The Swarm Grammar structure in Figure 7.7(a) inspired the photograph in Figure 7.7(b). The photograph is the result of an intricate experimental process: A glass plate was primed on one side. On the other side varnish was applied with a sponge in a subtle manner. Its palette was reworked with shades of white and blue. Undesired light reflections were removed that occur in photographs when working with varnish. The piece reflects the similarity between Swarm Grammar structures and clouds. Clouds, too, are spatial structures that continuously change.

The same process led from the Swarm Grammar structure in Figure 7.8(a) to the unnamed photograph displayed in Figure 7.8(b). Rhythmic collisions in the SG sculpture emerged through the synchronized differentiation into attracting and repelling swarm agents. The resulting structure motivated a competition between repelling color compounds. Black and white colors competed for surface space on a 17.3" x 25.2" glass plate. The tournament was performed by thinning the varnish to flow quickly with gravity. At the same time the compounds kept enough surface tension to repell each other.

Allowing color particles to find their own paths is very similar to the idea of autonomous Swarm Grammar agents leaving traces in space. To further this idea, the branching processes that led to the SG sculpture depicted in Figure 7.9(a) were emulated. A finish with a naturally stronger texture was used for the photograph shown in Figure 7.9(b). The coarse surface of a sloped wood panel propelled the color particles to disperse automatically. Moreover, the artist rotated the piece so that the color ran into different directions. Thereby, the artist only indirectly affected the outcome.

7.6.2 Finding the In-between World

In the works we present in this section, the Swarm Grammar sculptures were used as inspirational starting points for real sculptures. Then, in accordance with Reder's quote, the artist diligently chose the photographic configuration that led to the desired outcome.

The newly gained spatiality fully springs to life in the pieces in Figures 7.10(b) and 7.11(b). The interplay of light and shade creates deep, smooth textures, provoking tactile urges. The photographs convey organic looks and the fuzziness that we know from natural phenomena. They look soft and gnarly, like skin, but have their very own unique structures.

Figure 7.10(a) shows a Swarm Grammar structure with a compact, yet spiky core to the right-hand side and seven thick outgrowing, entangling ramifications. Their regular segmentation and their pointy ends capture the viewer's focus. Inspired by this virtual sculpture, the artist (T.W.) chose wires and tinfoil as construction elements for a corresponding real-world model. The decision for these materials was made to mimic the compact wrappings and ramifications of the SG sculpture. Although the original swarm structure is not directly identifiable in the photograph (Figure 7.10(b)), several analogies can be recognized. (1) The partition of the aperture: Few light elements on the left-hand side and impenetrable surfaces to the right. (2) Interwoven offshoots to the left and one protruding one in the top-right pane. (3) A rippling spiral similar to the segmented ramifications in the SG structure. Most importantly, however, very general, common features stand out. Firstly, the unity and the role of the interwoven elements. And secondly, the smoothness and the dynamics of the sculpture.

Searching for other suitable materials to simulate Swarm Grammar processes, the artist relied on paper for the piece presented in Figure 7.11(b). A soft light reflection is introduced through the material. The swirly twist of the SG agent illustrated in Figure 7.11(a) is directly reproduced in the sculpture in a manifold way. In this case a craggy, concave surface was devised. The radial gradients in combination with the fuzziness of the photograph create the impression of circular motion. The white background of the simulation screenshot was turned into black for the photograph, spotlighting the sculpture and allowing for an adroit illumination.

7.7 Summary & Future Work

We briefly outlined the idea of Swarm Grammars as a bio-inspired, computational developmental model. Seeking ways to explore the originality and novelty of this model, interdisciplinary work with a group of artists was performed. Several pieces of art inspired by Swarm Grammars were presented and discussed. We put an emphasis on the process of crafting the respective piece, its relation to the Swarm Grammar structure and new aspects that emerged because of the artistic discourse.

In particular, it was shown that rigid Swarm Grammar structures can be interwoven to open up virtual spaces (Section 7.4). The remarkable organic looks of Swarm Grammar structures create the impression of artificial but 'living' spaces, i.e., artificial inhabited worlds. Work has been presented that innovates on the transition between two-dimensionally mapped visualization and the third dimension by means of layering semi-transparent plates (Section 7.5). Dynamics and complexity of the underlying Swarm Grammar structure were maintained by allowing to rotate and overlay the painted swarm constructions. Finally, a series of pieces was presented that furthers the idea of modeling as an integral part of experimenting (Section 7.6). The chain from biological models to a computational meta-model to a structural instance that serves as model for artistic work grows once again. The artist used this long chain of modeling to mimic Swarm Grammar simulations from several perspectives. By means of a high-contrast painting method on semi-transparent glass plates, abstract features of Swarm Grammar constructions were captured. As a next step, the self-organizing properties of Swarm Grammar systems were emulated by repelling color compounds, dispersion through textured surfaces and swift rotations of the media to direct the flow of color by gravity. Different materials—wire, tinfoil and paper—were utilized to devise sculptures that gained soft, organic looks and also maintained other properties exhibited by the Swarm Grammar constructions.

Interdisciplinary work took place through the collaboration of scientists and artists. We believe that the discussed pieces are both an enrichment for the world of art and an inspiration to further the involved models on any level of abstraction—in silico, in vivo, or in-between.

For future work we want to improve on several concepts and approaches. The applied Swarm Grammar model is still restrictive in many ways: The assortment of construction elements is

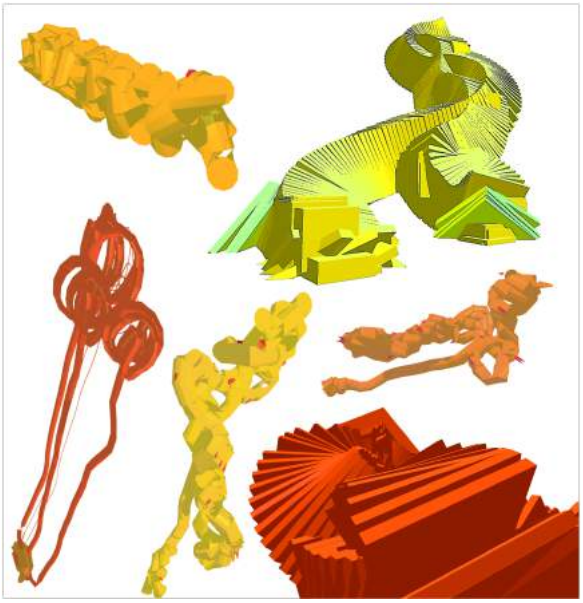
very small and little imaginative—broader selections and more intricate shapes could enhance the artistic value of the swarm-built structures. Due to the overwhelming number of parameters that determine a Swarm Grammar construction, the only, currently implemented way to realize a conception is through computational evolution. Although we believe in the power of this approach, a working artist expects greater freedom for individual arrangements in many cases. Therefore, embedding an interactive design tool into the Swarm Grammar simulation framework would be beneficial.

Regarding new concepts for Swarm Grammar art, we are currently developing mechanisms for the self-organized assembly of three-dimensional structures, following the idea of autonomous color particle paintings. In order to do so, the stepwise extension of the particles' attributes in tandem with means to affect the particles' situations and interactions needs to be investigated. As a starting point, we rely on layering propelled color particles—inspired by some of the early works by the Canadian artist Gerald Hushlak [411] and the sculpture "Pirouette in Red" (Figure 7.5) and the photographs seen in Figures 7.8(b) and 7.9(b). A series of photographs of experimental trials is displayed in Figure 7.12. Although this approach has similarities to modern 3D plotting devices [412] and feeds on real-world experiments on self-assembly [413], we believe that the swarm perspective provides a playground for novel ideas.

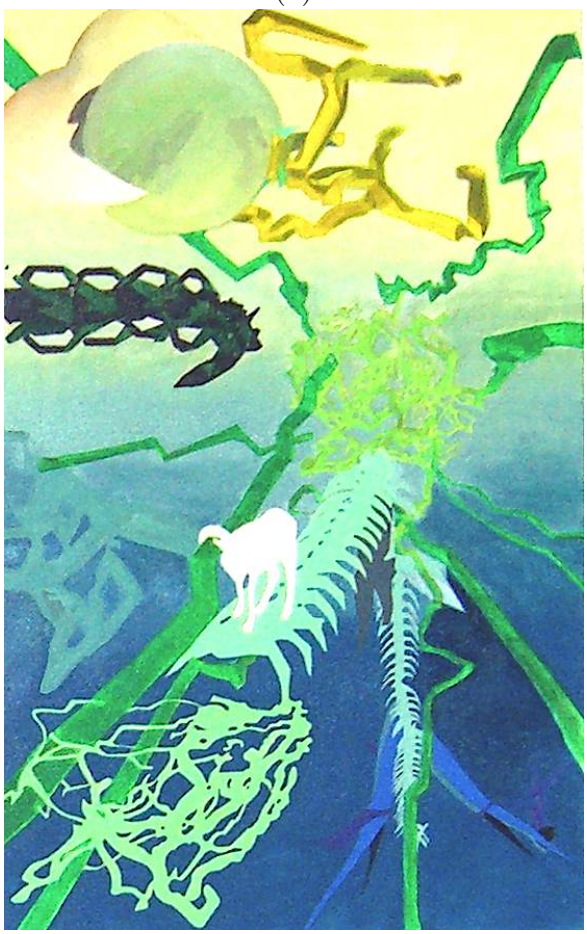
Acknowledgements

We would like to thank Professor Gerald Hushlak of the Department of Art of the University of Calgary for his continuing encouragement of our interdisciplinary work, his invaluable feedback and his logistic support.

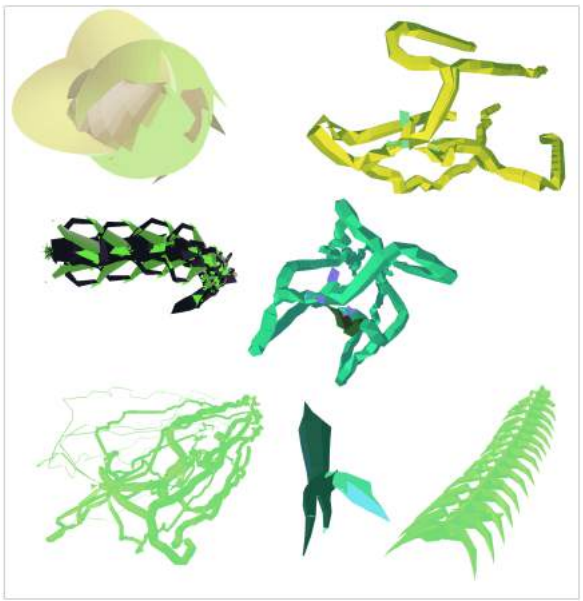
(a)



(b)

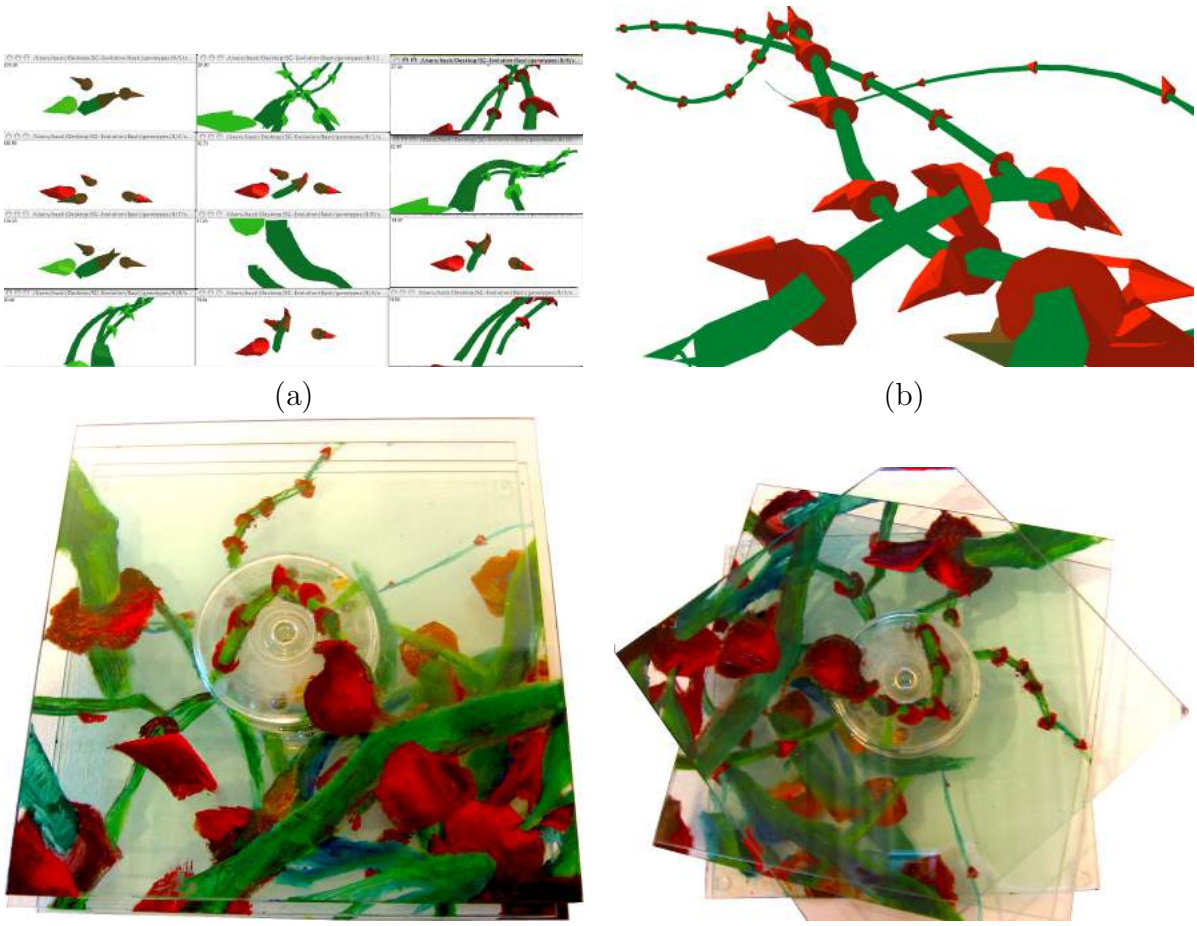


(c)



(d)

Figure 7.4: Diptych of the two pieces (a) “camlon” and (c) “bighorn sheep”. Acrylic medium on canvas, 23” x 38”. Selections of Swarm Grammar structures bred for the diptych are displayed in (b) and (d), respectively. (S.v.M., 2008)



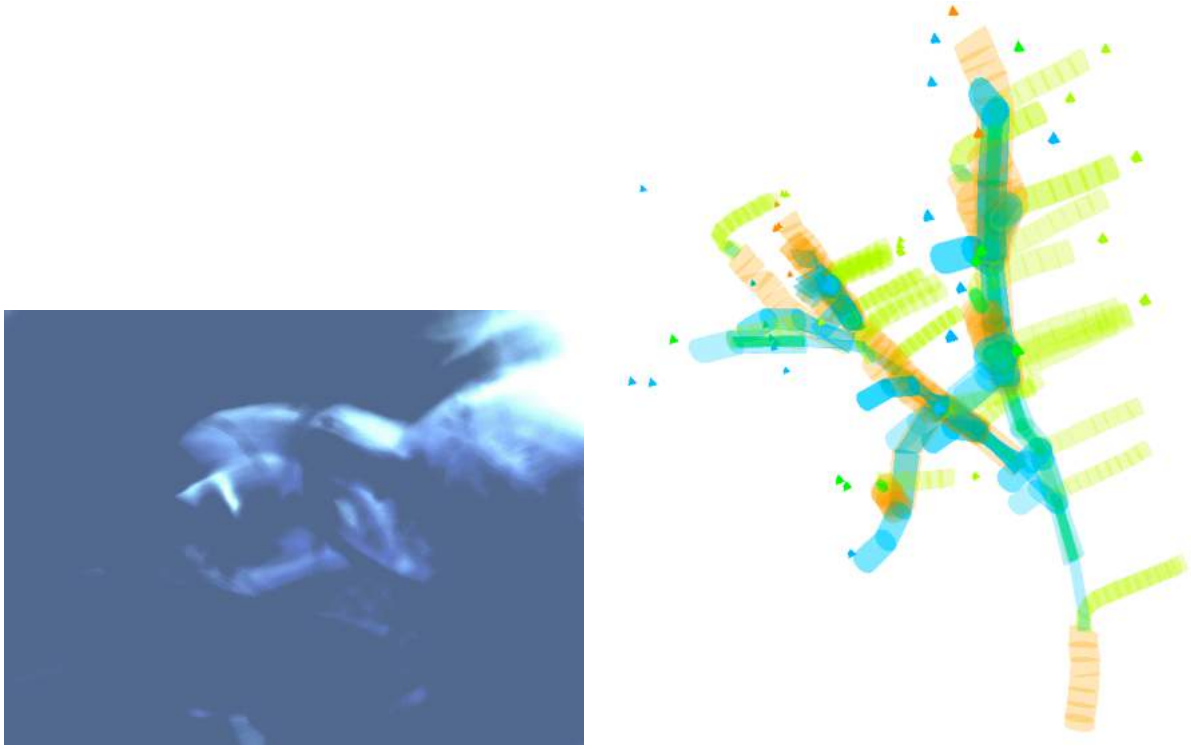
(a) (b)

(c) (d)

Figure 7.5: During an interactive evolutionary run (a), the depicted rose-like structure emerged (b) that inspired the piece: Pirouette in Red. 12" x 12", mixed media. (c) The sculpture in its initial position. (d) Bearings allow to rotate the plates into different configurations. (J.W., 2008).

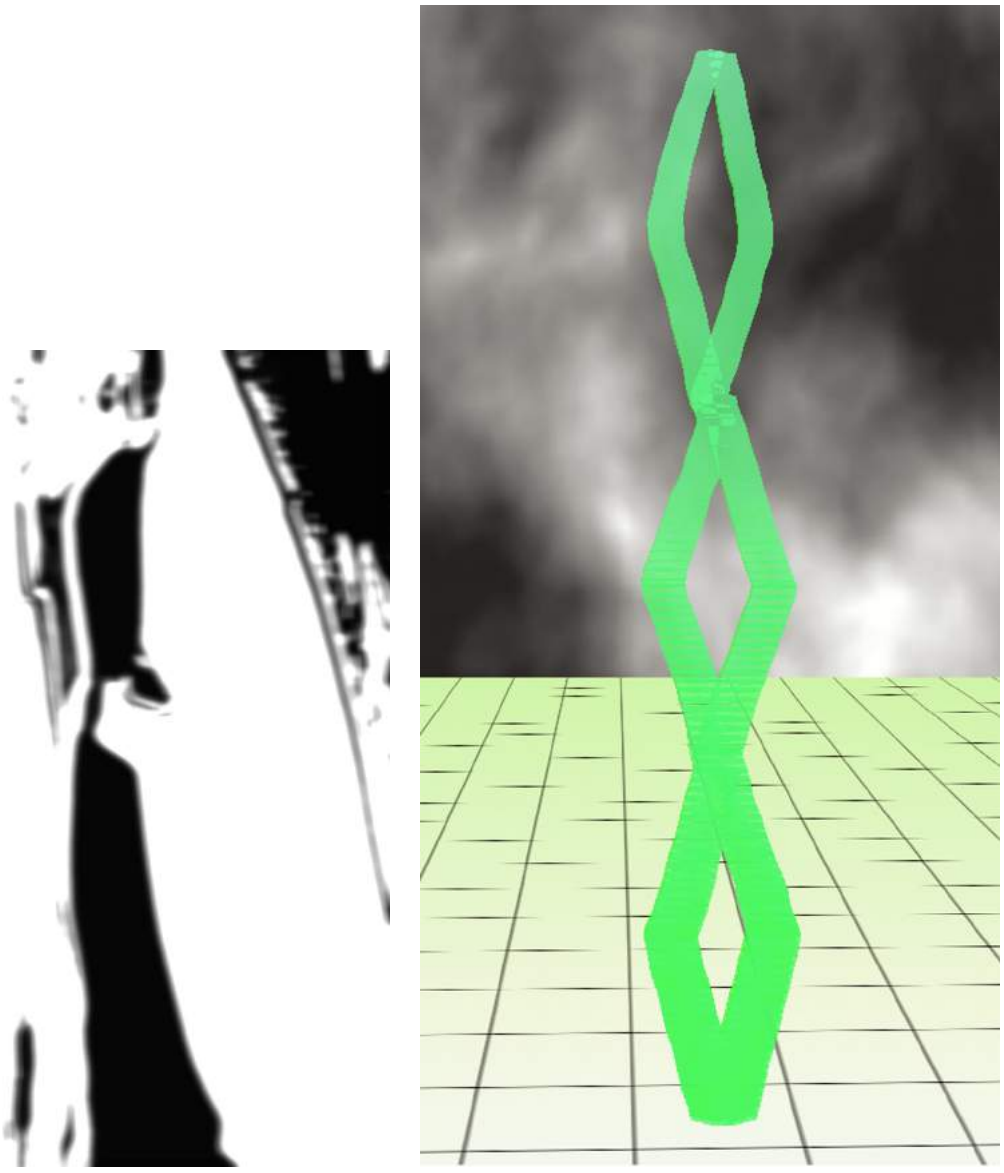
(a) (b)

Figure 7.6: (a) "Gate", photograph, varnish on a wood pannel (22.4" x 39.4"). (b) The inspiring Swarm Grammar sculpture. (T.W., 2008)



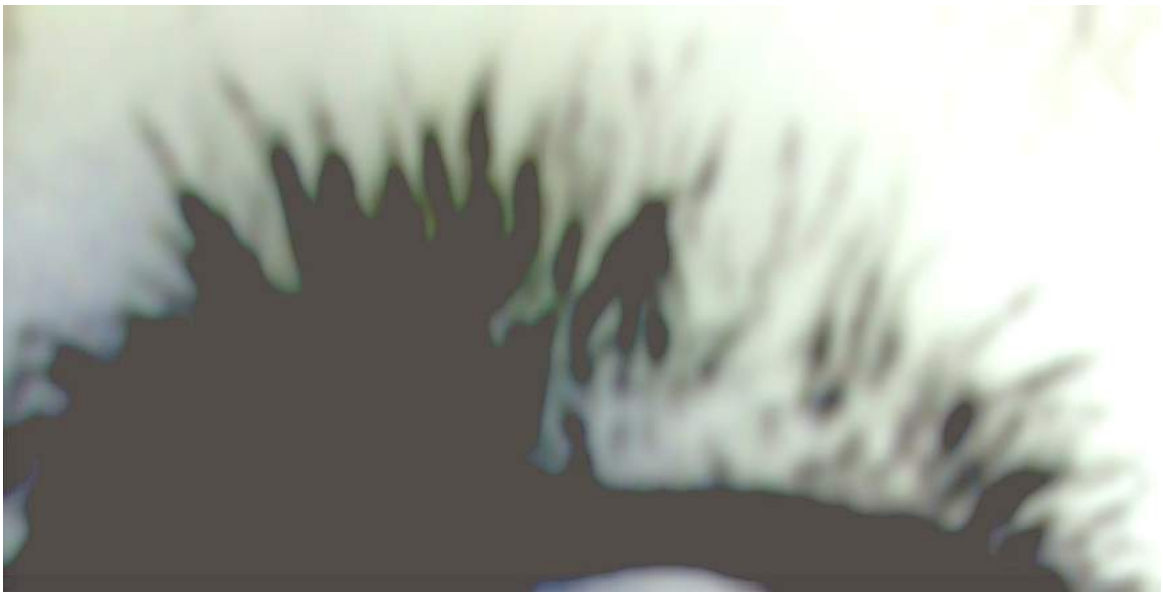
(a) (b)

Figure 7.7: Experimental processes involving painting with mixed media and reworked photographs led to (a) the arts piece "Cloud". It was inspired by the semi-transparent branching SG structure in (b). (T.W., 2008)

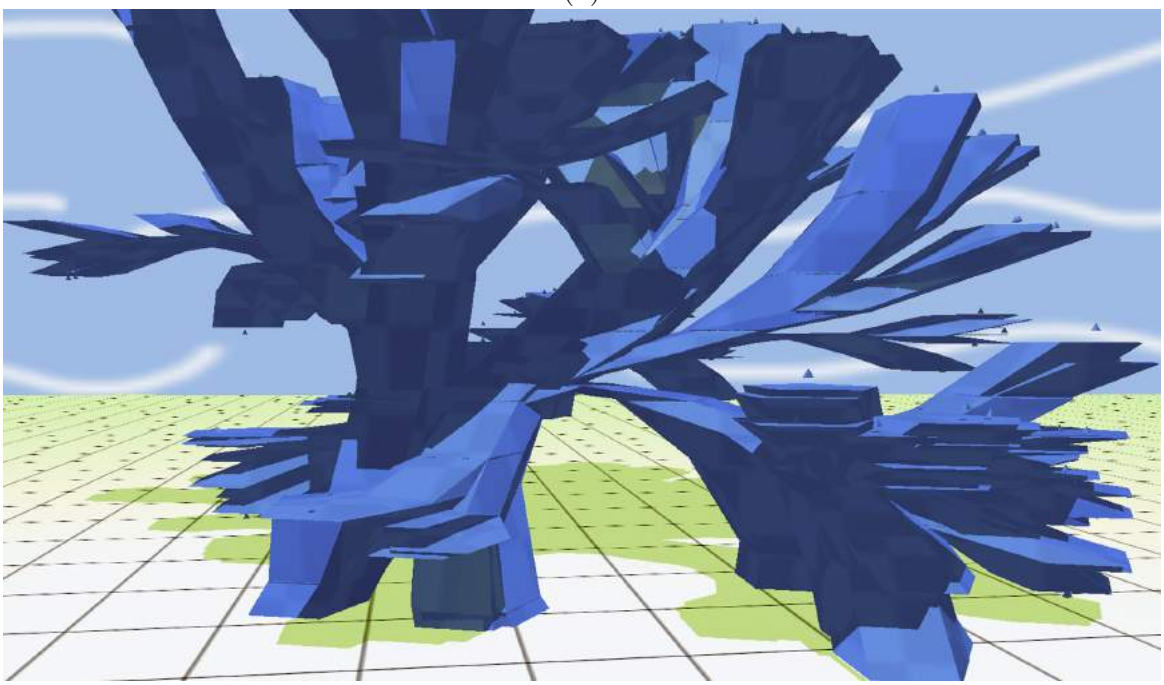


(a) (b)

Figure 7.8: (a) Black and white color compounds compete for surface space of a glass plate. As inspiration served the SG structure in (b), built by attracting and repelling swarm agents. (T.W., 2008)

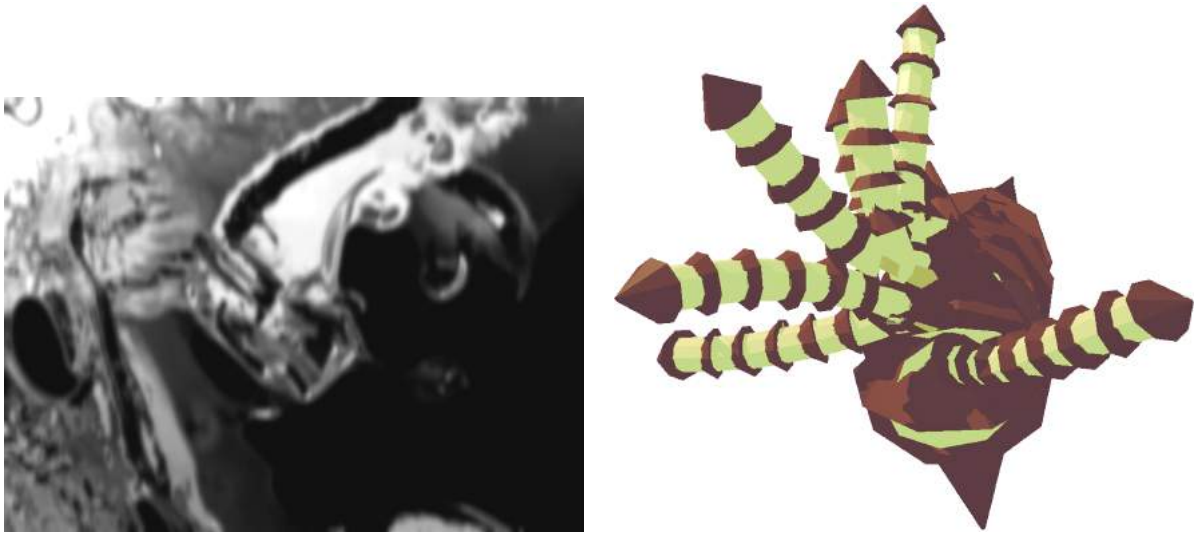


(a)



(b)

Figure 7.9: (a) "Blue": Color particles dispersed through gravity and the coarse texture of a 13.8" x 27.6" wood pannel. (b) The inspirational SG structure. (T.W., 2008)



(a) (b)

Figure 7.10: (a) Photograph of a sculpture made of wires and tinfoil with dimensions 7.9" x 9.4" x 2.8". Its inspirational Swarm Grammar structure is shown in (b). (T.W., 2008)

(a) (b)

Figure 7.11: (a) The attempt to form swarms with paper. Inspired by the Swarm Grammar graphic seen in (b). (T.W., 2008)

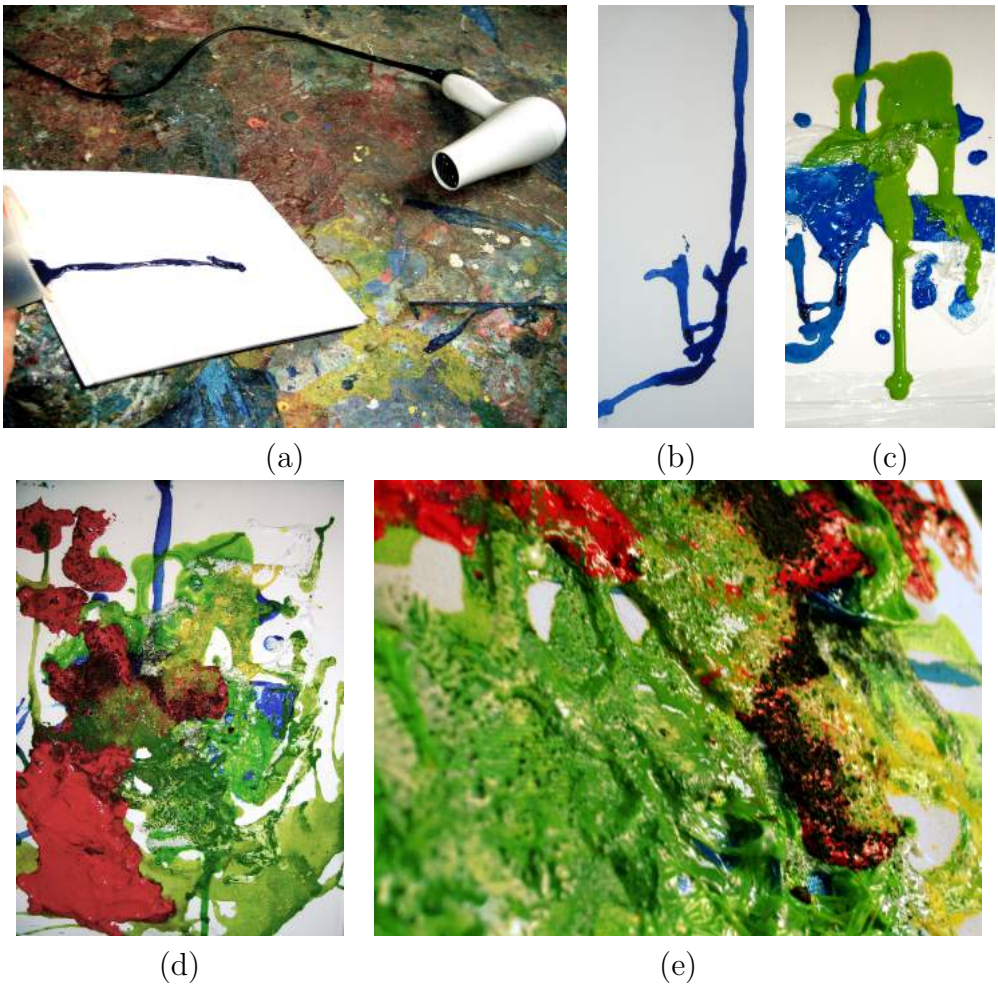


Figure 7.12: The displayed series of photographs shows the artistic reproduction of developmental swarm structures. (a) While color is running down a sloped canvas, a dryer works against gravity by blowing towards the induced current. (b) The interplay of forces creates branching structures. (c) New layers of color and plastic foil are introduced into the artificially created physical world. (d) Complex structural relationships have emerged through the interacting layers of mixed media. (e) The emergent texture has grown into three dimensions. (S.v.M., 2009)

Chapter 8

The Digital Aquarist: An Interactive Ecology Simulator

In this paper, we present an interactive simulation of the ecological cycle in a fish tank. Like the owner of a real fish tank, the user of the simulation has to balance several vital parameters of the aquatic system. Next to people interested in the world of aquatics in general, the simulation especially targets teenagers and aims at increasing their interest in ecosystems, and to contributing to their understanding of basic ecological principles. We engage the user introducing various gamification elements, including game-like UI elements, high scores, and a diligently adjusted reward system that allows for adding new inhabitants to the aquarium. Based on a user study, we evaluate our concept and layout possible improvements and extensions.

Julian Schikarski, Oliver Meisch, Sarah Edenhofer, **Sebastian von Mammen**: *The Digital Aquarist: An Interactive Ecology Simulator*. In: *Proceedings of the European Conference on Artificial Life 2015 (ECAL)*, 2015, pp. 389–396.

8.1 Introduction

A profound understanding of complex relationships and processes in ecological systems is an important factor for making informed, sustainable decisions. Like other empirical sciences, ecological research heavily relies on digital tools, including those for storage/retrieval, modelling and analysis ([414]). Educators have been assembling a similar repertoire of digital tools for

teaching ecological systems, whereas computational simulations are especially well suited to convey their inherent, often fragile complexity ([415]).

In this paper, we present *The Digital Aquarist*, an interactive simulation of the ecological cycle in a fish tank. Following the concept of gamification, see e.g. [416, 417], we engage the user in the simulation by providing easy and rewarding access to the model context, to the model mechanics and especially to the offered user interactions. Along the same lines, we reduce the amount of prior knowledge required for a rewarding simulation experience to a bare minimum: (1) The user can easily explore the interdependencies between different organisms by himself. (2) The simulation is staged in a moderately sized fish tank that is often found in private homes (about 6.5% of households keep fish in the U.S. ([418])).

We further distilled a model complex enough to convey foundational ecological relationships and the emergent system dynamics, yet simple enough to work for an introductory educational setting. The interaction possibilities are part of this model simplification: The user is encouraged to balance the different system variables by adding or removing organisms from the fish tank. Each animal or plant has a certain impact on the ecosystem by either reducing or increasing systemic parameter values, e.g. through breathing. The goal is to keep the schools of fish healthy over a long period of time while increasing the number of inhabitants, and thus the heterogeneity and the complexity of the ecosystems' population.

The remainder of this paper is structured as follows: In the next section, we discuss related work that influenced this project. Afterward, we present our modelling approach, the use of gamification elements, the realisation of aesthetic visualisation and the degree of complexity of the application. Based on our approach, we summarise our accomplishments. Finally, we outline how the simulation could be extended in the future.

8.2 Related Work

Creating a closed ecological cycle has been attempted by scientists in various projects. Biosphere 2 is a notable representative project of this kind ([419]). It is an architectural and technological large-scale compound for exploring the interplay between human life and its environment in a closed ecological system. Biosphere 2 had originally been planned as a self-

sustaining system. It is used as a research laboratory now, after two attempts to make it work have failed. The same idea but, at least commercially, more successful is the Ecosphere, an aquarium whose inhabitants are completely sealed off from any metabolic exchange with the environment. Only the sun light enters from the outside world and drives growth and transformation processes of the contained plants and animals ([420]). Although it is being disputed whether the life stock, the shrimp Halocardina Rubra, is surviving rather than slowly starving to death ([421]).

Despite their minimalistic approaches and their emphasis on the exploration of well-defined ecological processes, neither Biosphere 2 nor the Ecosphere are apt for learning about and exploring ecosystem dynamics. One reason is the inaccessibility of the given systems. It is barely possible to change any of their compositions. Time scales are another reason: It takes long periods of time for ecological systems to stabilize, rendering it (even more) impractical to proactively change their settings and to explore their dynamics. For these reasons, providing interactive simulations of isolated problem domains, or *microworlds*, has become an important methodological approach in education and education research ([422, 423]).

While it has been emphasised that idealised model representations (as opposed to concrete ones) foster the development of generalisable insights ([424]), *The Digital Aquarist* prioritises relatable, engaging aesthetic animation over the abstract display of an expectedly vivid, visually attractive ecosystem. Therefore, different from a preceding, NetLogo-based 2D aquarium model ([425]), *The Digital Aquarist* models the aquarium and its inhabitants in an animated three-dimensional world.

Interactive parameter adjustment of a simulated aquarium has previously been used to study human learning and planning capacity ([426]). It could show that promoting the free exploration of a dynamic system allows the user to gain general knowledge, whereas addressing specific tasks would solely foster specific knowledge. We harness this insight by allowing the users to freely explore our simulation and to only provide implicit stimuli to maintain the aquarium over time and with growing heterogeneity. However, as the investigation of intellectual capacities is not the *The Digital Aquarist*'s goal, we also reveal detailed information about the inhabitants of the simulation and their relationships, if inquired by the user.

8.3 Methodology

For the sake of accessibility, we focus on simple visuals and avoid overburdening the user with information which would, very likely, jeopardise the attractiveness of the simulation ([103]). Instead of promoting formal analytical skills, we make sure to provide visible feedback similar to real-world experiences, including rampant algae growth upon eutrophication or starving fish. To keep the user both interested and involved, we built the simulation on the three "pillars of fun" ([427])— relatedness, competence, and autonomy: (1) Relatedness is established by the fact that aquaria may exist in households similar to the ones the potential users of the simulation call their home (see Figure 8.1). The great number of aquaria worldwide renders it likely that the users are even familiar with the concept of keeping ornamental fish, possibly also the notion that the aquarist needs to ensure an ecological balance. Finally, we establish a connection between the user and the simulation by showing that initially the virtual aquarium is empty and thus that he is responsible for each and every one of its inhabitants. (2) To promote the competence of the user, he needs to be challenged without giving rise to frustration. This goal is supported by the facts that *The Digital Aquarist* builds an ecosystem one step at a time, that the user always has the power to change its configuration back to a previous, simpler state, and that he can pro-actively inquire information about the aquarium's inhabitants and their relationships. To ease the user into the simulation scenario, he may enter a tutorial level from the main screen (Figure 8.2) and step through a guided tour shedding light on the impact of different species on the ecosystem. (3) *The Digital Aquarist* provides an inherently autonomous user experience in that it does not enforce the fulfilment of specific tasks but it lets the user explore the aquarium dynamics on his own. A high score system is provided that rewards the user's achievements but it does not limit the potential of exploration.

8.3.1 The Aquarium Model

In order to ensure an ecological equilibrium, several variables that describe a fish tank's state have to be maintained at certain levels. The main parameters are the levels of oxygen, carbon dioxide, nutrient matter in the seabed, the water volume, and the unoccupied space in the tank. Other important factors are the hardness of the water, its temperature, and the tank's light exposure. To allow the user to focus on key aspects of the ecological cycle, the latter two

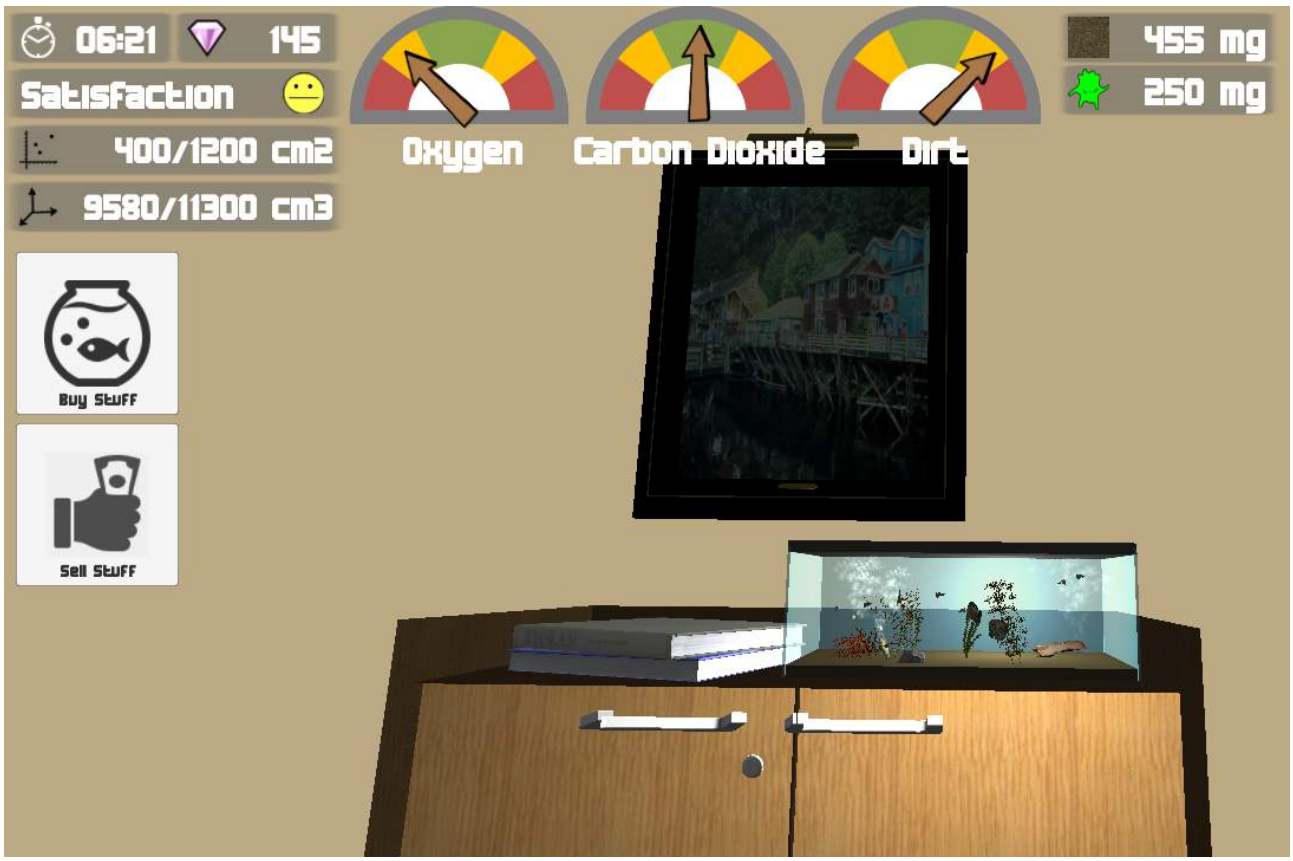


Figure 8.1: The simulated aquarium placed on a cupboard signals an everyday real-life scenario. The user interface aligned at the border of the view invites the user to join a playful simulation session.

Figure 8.2: From the main menu of *The Digital Aquarist*, the user may access the high score list, enter a tutorial or join an endless explorative simulation session.

aspects are neglected in our model. In a real aquarium, these factors have to be adjusted by means of external devices.

The user can balance the aquarium parameters by adding and removing various animals and plants which all have their own way of interacting with the system. The amount of plants impacts the amount of nutrient matter in the water and the seabed, the levels of oxygen and carbon dioxide in the water, as well as the amount of unoccupied space in the tank. Seaweed breathes in carbon dioxide and breathes out oxygen, takes nutrient matter out of the seabed and releases small particles of nutrient matter into the water.

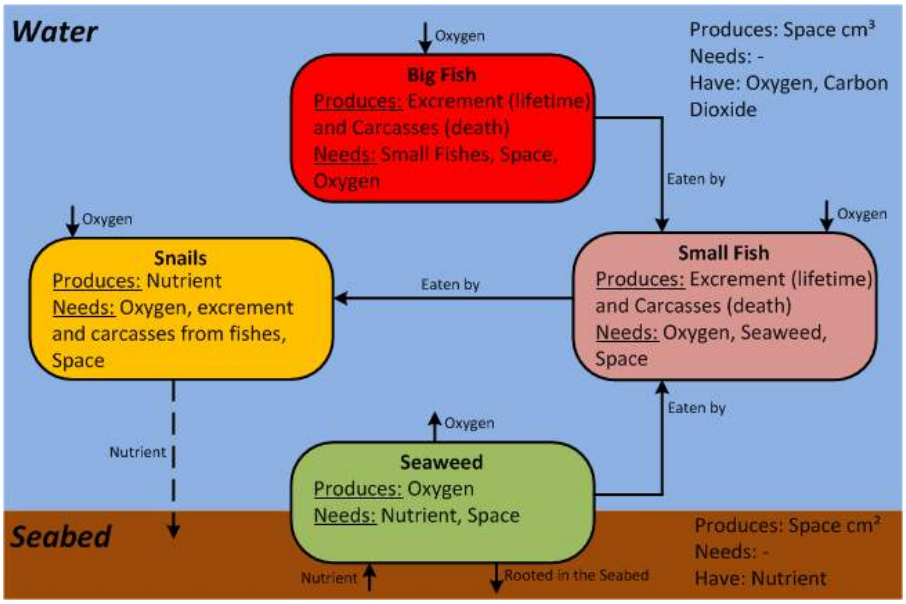


Figure 8.3: Overview of our fish tank ecosystem model.

Figure 8.3 provides an overview of the inhabitants of the aquarium and their impact on the ecosystem. Nutrient matter in the water is consumed by the fish. The snails add the fish' excrements to the seabed. The seabed in turn serves as a nutritional basis for the seaweed. Small parts of the seaweed that break away and enrich the water are picked up by the fish again. The seaweed also produces the oxygen snails and fish breathe and absorbs the carbon dioxide they produce. This cycle can be disrupted by fish either eating smaller peers or seaweed in great quantities, which happens if there is not enough nutrient matter in the water.

Organismal Interdependencies

The food intake of the fish scales with their size/age. Next to fish and plants, snails populate the virtual aquarium. Both micro-organisms and snails transform the fish' excrements in the water

in nutrient matter that agglomerates in the seabed. This mechanism completes the nutrition cycle in the system. In order to provide ample visibility, the presence of snails also represents the transformative power of micro-organisms in our simulation. This means that snails are the only organisms accessible to the user that filter dirty water (from fish excrements) and feed nutrients into the seabed. In reality these processes would be addressed by both snails and micro-organisms. Same as fish, snails breathe in oxygen and breathe out carbon dioxide.

For the user to create a closed ecosystem, he needs to add members of each class of organisms and ensure that their mutual impacts keep a nice balance. Fish and snails need to breathe in a certain amount of oxygen and need the carbon dioxide to be below a certain level to keep from suffocating. Plants need to breathe in a certain amount of carbon dioxide, otherwise they suffocate. Fish need to absorb nutrient matter from the water, snails filter the fish' excrements, and seaweed absorbs nutrient matter from the seabed.

Additionally to the aforementioned interdependencies, each organism takes up a certain amount of space. When the fish tank gets too crowded, the fish will get stressed and will be unable to procreate. The procreation of fish adds another layer of complexity to the system. Due to the procreation of the fish, the user cannot easily anticipate the needed amount of inhabitants of the aquarium before starting the simulation. Therefore, the user needs to react to changing conditions on the fly, either by removing individual fish or by providing more nutrient matter, as well as snails and plants to find a new equilibrium.

Model information about the individual organisms is made available to the user on demand. A shopping interface allows to choose and add organisms to the ecosystem. The user needs to earn virtual currency to buy the organisms in the store. He earns coins by keeping animals and plants alive for as long as possible. This positive feedback mechanism ensures that the user is not immediately overwhelmed by a great number of organisms in the tank and also that fewer expensive organisms are added at first that are harder to cope with. At the same time, keeping a healthy ecosystem is directly translated into a rewarding sensation with actual impact on the interaction possibilities.

Currently, a selection of five fish is offered in the virtual store. Their impact on the ecosystem only differs due to their different sizes which in turn affects their metabolic rates. Otherwise, they all play the same role in the system. That means that all fish breathe in oxygen, consume nutrient matter, emit carbon dioxide, and leave excrements behind. Yet, the respective amounts

vary from species to species. The store interface also provides additional information about the organisms, as exemplarily shown in Figure 8.4.

Figure 8.4: Additional information about a guppy fish is offered to the user in the virtual shopping interface.

Modelling Metabolisms

There are several model assumptions that have been made to keep the model complexity manageable. In particular, we assume a constant water temperature of $20°C$, we do not consider the day/night cycle, we consider $9mg/l$ of oxygen as fully saturated freshwater ([428]), $50mg$ O_2 consumption per $100g$ of fish body weight per hour ([429]), $5mg$ O_2 production by $1g$ of algae per hour, and a $10sec$ integration step size of the simulation.

Table 8.1 lists the model variables involved in the calculation of the degree of oxygen saturation in the tank. The binary function $oCO_2(t)$ indicates whether or not a surfeit of carbon dioxide can be determined at time step t, i.e. a CO_2 value greater than or equal to twice the standard level of CO_2 is detected.

$h, w, d \in \mathbb{R}^*_+$	fish tank dimensions (cm)
$c \in \mathbb{R}^*_+ = h \times w \times d$	fish tank capacity (cm^3)
$li \in \mathbb{R}^*_+ = c/1000$	fish tank capacity (litres)
P	set of all plants
A	set of all animals
$O_2(t) \in \mathbb{R}^*_+ = li * 9$	amount of O_2 (in mg) at time $t = 0$
$i = 10$	integration step size (seconds)
$oCO_2(t) \in \{0, 1\}$	strong over saturation of CO_2 at time t

Table 8.1: Model variables for calculating oxygen saturation.

Based on the given variables, we calculate the amount of O_2 (in mg) at time t according to Equation 8.1. The oxygen saturation level is the ratio of current oxygen in the system relative to the initial oxygen level at $t = 0$. In conclusion, the oxygen saturation is influenced by the amount of oxygen produced and used by each organism in the fish tank. Exemplary values for the O_2 consumption of model organisms are listed in Table 8.2, whereas the intake is negative for algae since they produce oxygen.

$$\begin{aligned} O_2(t) = O_2(t-1) + \sum_{p \in P} O_2(p) - \sum_{a \in A} O_2(a) \\ - (\sum_{a \in A} O_2(a) * 0.25) * oCO_2(t-1) \end{aligned} \tag{8.1}$$

species	body weight	O_2 intake
Pterophyllum scalare	$300g$	$0.42\ mg/i$
Poecilia reticulata	$10g$	$0.014\ mg/i$
Aponogeton ulvaceus	$50g$	$-1.39\ mg/i$
Alternanthera reineckii	$100g$	$-2.78\ mg/i$

Table 8.2: Exemplary values of O_2 intake of model organisms.

Figure 8.5 shows an exemplary evolution of the oxygen saturation level. Until $t = 4$, 24 fish (12 *Pterophyllum scalare* and 12 *Poecilia reticulata*) slowly decrease the level of oxygen in the aquarium, despite the presence of two plants (*Aponogeton ulvaceus* and *Alternanthera reineckii*). At $t = 4$, one of the two plants dies off and the oxygen depletes twice as fast as before. The lack of oxygen leads to suffocation of 21 fish at $t = 7$ which results in the recovery

of the oxygen saturation rate based on one remaining plant.

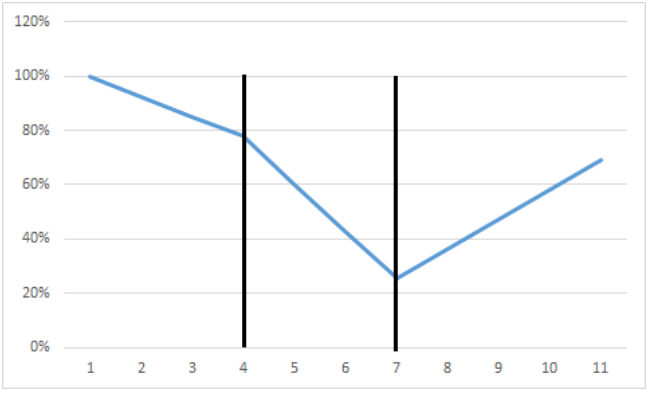


Figure 8.5: Oxygen saturation over time.

The amount of nutrients, the amount of dirt and the amount of carbon dioxide are computed in the same way as oxygen. Yet, the according equations consider the different organisms' impact on these variables. In case of nutrients in the water, the organisms take on the same role as for the oxygen level—plants increase their level, animals reduce it. Only the actual values of nutrient matter provided and consumed differ. The roles of the organisms are switched in terms of carbon dioxide, i.e. animals exhale CO_2 output and plants consume it. Dirt arises from the set of all fish F and is diminished by the set of all snails S, resulting in Equation 8.2.

$$Dirt(t) = Dirt(t-1) + \sum_{f \in F} Dirt(f) - \sum_{s \in S} Dirt(s) \tag{8.2}$$

Simplifications

Balancing complexity and accessibility, we have setup an approximative model. Therefore, we have to investigate the impact on the model accuracy conveyed to the user. The fish offered to the user to populate the aquarium resemble two popular ornamental species, the guppy and the scalare. The guppy mainly feeds on zooplankton which is living plankton, in the simulation it feeds on plankton produced by seaweed ([430]). Scalare are known to eat small fish as portrayed in the simulation ([431]). Snails do have a cleaning effect on the fish tank, yet they usually eat leftover food and algae ([432]). As mentioned before, the snails also visually represent the role of micro-organisms in the ecosystem, to empower the user to easily trace and influence the delicate dependency network.

Despite its simplifications, the current model conveys foundational interdependencies an aquarist needs to be aware of. Therefore, we feel the overarching goal of educating about ecological systems' dynamics is not weakened. Yet, we would like to identify and integrate new ways of high-level visualisations for improving the model accuracy without jeopardising *The Digital Aquarist*'s accessibility.

8.3.2 User Challenges

The learnings of *The Digital Aquarist* result from freely exploring, learning (primarily) by trial-and-error, and mastering a potentially great complexity of an ecosystem. They include a notion of the basic interdependencies of the interacting species as well as their evolution over time: Depending on the metabolic status of the aquarium and the configuration of its population, the effect of adding individual organisms to or removing them from the tank is delayed. In order to successfully manage the aquarium, the user has to anticipate these developments. This is especially challenging as each organism influences more than one system variable.

Without user intervention, the collapse of the ecosystem might accelerate, for example by hungry guppies eating seaweed as seen in Figure 8.6. Devouring seaweed further lowers the amount of plankton in the water, thereby making food an even scarcer resource. This example illustrates how easy it is to disturb an ecologically balanced system and that restoring that balance is not easy, especially if many different species are involved.

8.3.3 User Interface

As seen in Figure 8.1, three system variables (O_2, CO_2, and dirt) are represented by gauges which enable the user to check the current levels at a glance. Coloured segments indicate the criticality of the respective variables, whereas green indicates a favourable situation, yellow requires the user's attention and red underlines a fatal system state. An increasing level of dirt is also reflected by the water gradually turning green (Figure 8.7). The icons on the left-hand side of the screen indicate the duration of the simulation in progress, the cumulative score and the overall satisfaction of all organisms in the tank. These information are represented by the timer, the diamond and the smiley icon, respectively. The levels of nutrient matter in

Figure 8.6: Guppies eating seaweed due to a lack of plankton in the water.

the seabed and water are displayed as numbers on the right-hand side of the screen, as the only restriction for them is not to reach zero. Since the fish tank provides limited space it is important for the user to know how many more fish and plants he can add to the system. Therefore on the lefthand side there are indicators how much space in cm^2 is left for plants in the seabed and how much space in cm^3 is left in the water for fish. The user interface enables an intuitive understanding of the status quo and quickly provides feedback about the ecosystem's evolution.

It is important to invest effort into the visual appeal of an interactive aquarium simulation—after all, ornamental fish are not only kept for the fascination for living organisms only but also for their elegance and beauty. Figure 8.8 shows the flocking of fish which mimics a life-like behaviour and a realistic look of aquarium. The flocking behaviour of fish was implemented according to the boid concept by [1]. Here, each individual moves in accordance with its neighbours (Figure 8.9). In particular, it is urged to keep a minimum distance from its peers (*separation urge*), to flock towards the average location of its neighbours (*cohesion urge*) and to align its velocity with their average velocity (*alignment urge*). We generate circular waypoints throughout the aquarium to let the schools' movement appear naturally.

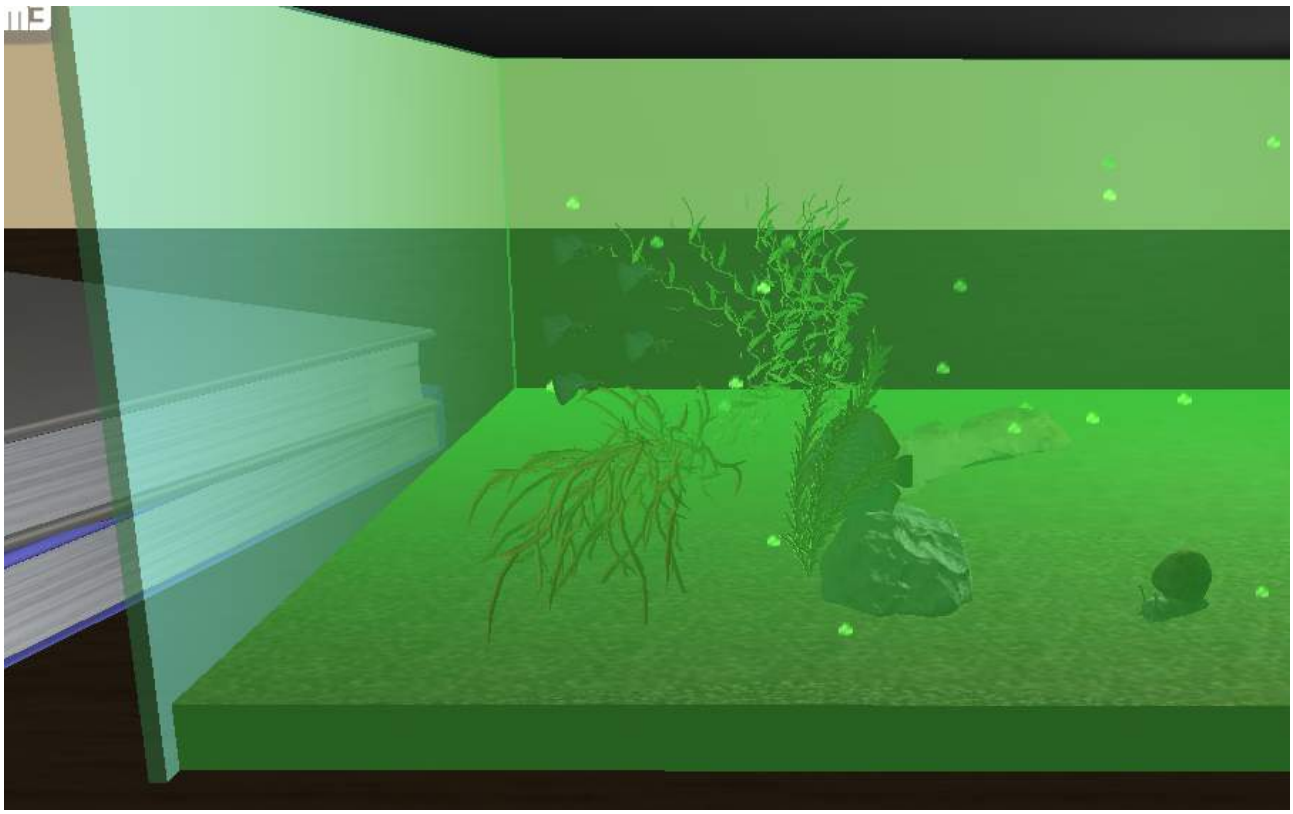


Figure 8.7: The water gradually turns green with an increasing degree of dirt.

Soothing background music creates an inviting, relaxing atmosphere. Typical sounds of aquarium pumps and occasional oxygen bubbles popping on the surface help the user feel immersed into the simulation.

8.4 Discussion

The user can effectively balance the system variables by adding and removing organisms to and from the aquarium. He is rewarded for using more complex scenarios by a scoring/virtual currency system. In particular, higher scores are achieved when hosting bigger fish like scalares rather than relatively small guppies. The earned points can be spent on further additions to the aquatic ecology. We made *The Digital Aquarist* available online and invited colleagues and acquaintances by means of email lists and social media postings to evaluate it. In the according online survey, 31 testers provided anonymous feedback.

Table 8.3 shows their ratings regarding general aspects of *The Digital Aquarist*. From left to right, the percentages of testers reflect which aspects were “very poor, poor, fair, good, or

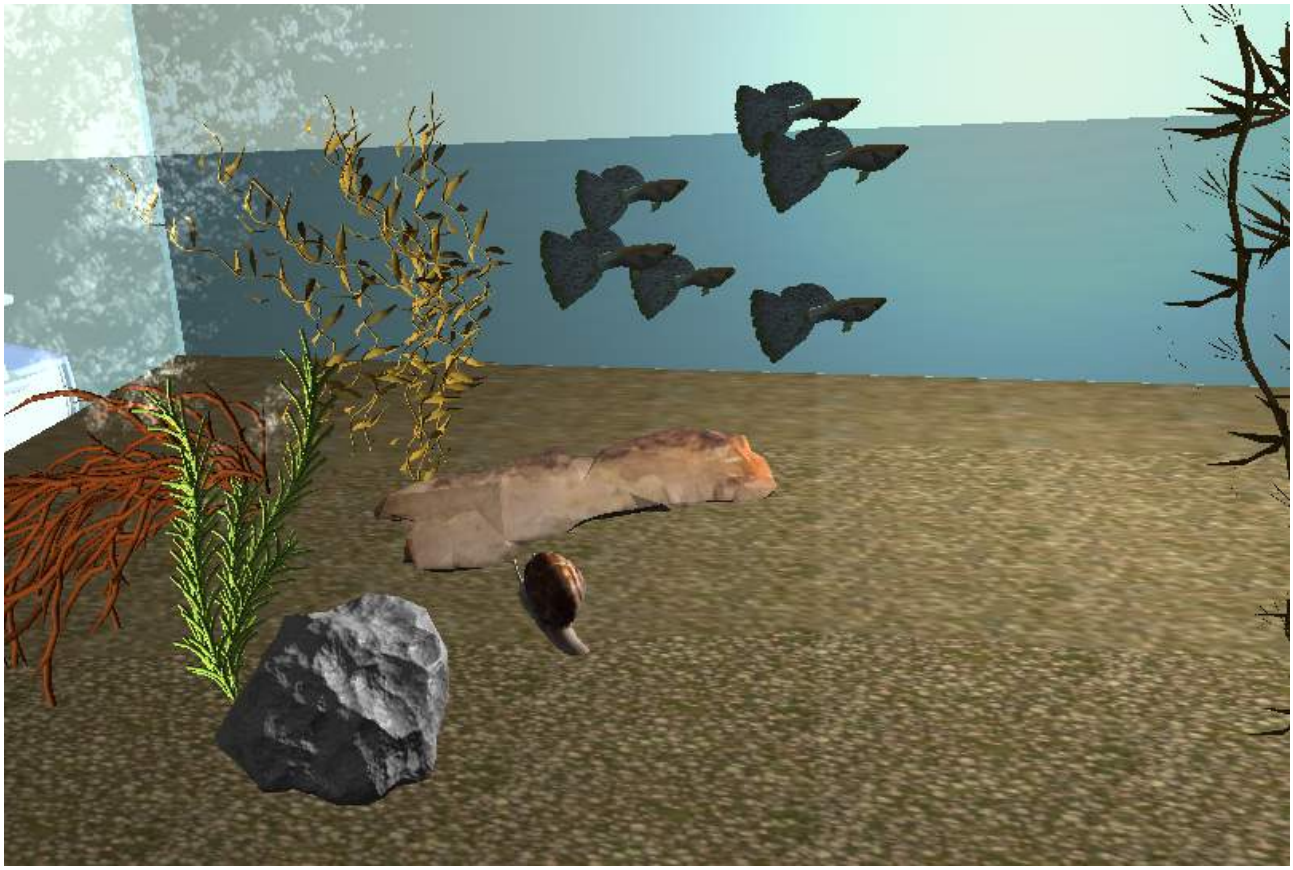


Figure 8.8: A school of guppies animated in accordance with the boids model ([1]).

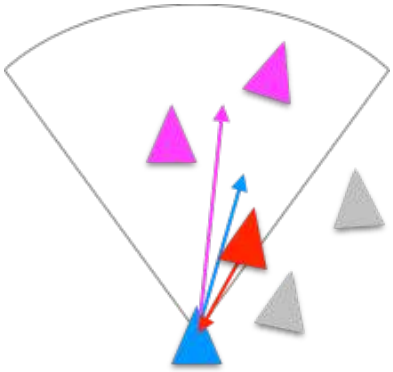


Figure 8.9: The boid flocking model considers cohesion towards perceived neighbours (pink arrow), separation from peers that are too close (red arrow) and alignment with the neighbours' average velocity (blue).

excellent" (represented as "$--, -, o, +, ++$" in the table). A majority felt that the topic of the game was a good choice, that the model complexity was appropriate, that *The Digital Aquarist* was easy to use and provided some fun. The aesthetics of the game and the learning effect were mainly rated as "fair", the intuitiveness of the game mechanics was rated as "poor". The latter fact stroke us as particularly interesting as the game mechanics are aligned with the model facts the users would learn—if they were considered intuitive in the first place,

there would be little knowledge that could be learned. And indeed, a majority of testers felt that they had learned about ecological balance (44, 44%) and about aquarium ecologies in particular (48, 15%). These opinions were supported by some multiple choice questions that inquired about the aquatic organisms' interactions. A great majority of testers recognised facts about the metabolism of fish, snails, and seaweed. Yet, their feeding habits were not as clearly understood. For instance, 11% of the testers erroneously thought snails contributed to the pollution of the water, only about 40% realised that fish eat other fish (which only happens if other food sources become scarce), and only 26% recognised that seaweed was involved in nutrient production.

	−−	−	*o*	+	++
Game Topic	0	12,9	25,81	**45,16**	16,13
Aesthetics	3,23	6,45	**38,71**	35,48	16,13
Model Complexity	0	12,9	32,26	**48,39**	6,45
Fun	16,13	16,13	25,81	**29,03**	12,9
Learning Effect	12,9	12,9	**41,94**	25,81	6,45
Intuitiveness of Game Mechanics	3,23	**35,48**	19,35	25,81	16,13
Ease of Use	6,67	10	20	**40**	23,33

Table 8.3: Ratings (in %) of different aspects of *The Digital Aquarist* provided by 31 anonymous testers.

8.5 Summary & Future Work

The Digital Aquarist provides a small-scale ecosystem based on simplified metabolic models. An accessible user interface is supported by animation, visualisation and audio tracks to provide for an open-ended simulation experience that conveys the delicate balance needed to maintain a complex system.

A user survey of our first implementation of *The Digital Aquarist* indicates that we have successfully addressed certain challenges, including finding a proper level of abstraction of the simulated model as well as providing the necessary accessibility. Yet, we also appreciate that there is leeway for further improvement. Component-based development environments render

it quite easy to setup intricate tracking shots that could allow the user to follow individual fish or snails, to experience the ecological processes in a more immersive manner and to reveal interactions close-up. These perspectives could be supported by intricate animations, for instance based on particle systems, to clearly visualise organismal activities such as nibbling, eating, and excreting. In addition, diagrammatic augmentation could drastically speed up the learning process, indicating the relationships among the organisms on demand. In order to keep the user interested over a long period of time, the repertoire of available species, decorative items, and technical add-ons could grow after successfully mastering a balance for a given timespan. Along these lines, one should even consider providing different sizes, shapes and kinds of aquariums. The iconic fishbowl bears different possibilities and challenges than a saltwater tank.

We have scheduled a demo/play event for teenagers, our targeted user group. Based on its success, we are planning the public, mobile release of *The Digital Aquarist*.

Chapter 9

PowerSurge: A Serious Game on Power Transmission Networks

In this paper, we present an interactive serious game about power transmission systems. The system familiarizes novices with the basic design and behavior of such systems. Using simple drag and drop interactions, power plants and consumers are placed and connected in a virtual landscape that is presented from an isometric perspective. A series of tutorials fosters the user's mastery in building and controlling a complex system. The advanced user is challenged by tasks such as the redesign of an established power infrastructure to integrate a large percentage of regenerative power plants. Next to the interface, we detail the model that drives the simulation. The methodologies presented in this paper can be applied to a wide range of serious games about complex network designs.

Sebastian von Mammen, Fabian Hertwig, Patrick Lehner, Florian Obermayer: *PowerSurge: A Serious Game on Power Transmission Networks.* In: *EvoApplications Proceedings of the 18th European Conference on Evolutionary Computation (EvoStar)*, vol. 9028, Springer, April 2015, pp. 406–417.

9.1 Introduction

Energy management is an important challenge that governments have to struggle with. The general trend to turn away from climate straining and unsafe technology such as nuclear power

and fossil fuels to smaller but renewable energy sources requires a more decentralized energy distribution infrastructure [433]. The turn from all-time available power to regenerative but unsteady resources poses non-trivial challenges: Energy has to be efficiently stored locally to counteract the sporadic absence of wind and solar radiation, see for instance [434] and [435]. On the other hand, transportation of electricity from regions with high yield has to happen efficiently. Adapting the infrastructure is a difficult task, and it is useful to explore the existing capabilities and virtualize any changes before their implementation. A broad overview addressing the challenges of current Power Transmission Systems (PTS) is provided by [436].

In this paper, we present PowerSurge, an interactive simulator for PTS. We detail its user interface, gamification elements and the underlying model that drives the simulation. PowerSurge introduces users who are new to the field of PTS to gain a high-level understanding of the challenges faced in their design and maintenance, and a feeling for complex behaviors in such networks. Due to the similarity to other complex science themes such as social or economic systems, see for instance [437], the methodologies presented in this paper can be transferred to a wide range of serious games.

In Section 9.2, we first survey existing power simulation systems, emphasizing their distinctive features. Next, in Section 9.3, we present our software, including its design principles, its user interface, and a detailed discussion of our domain model. We conclude with a short summary of our results and an outlook on future work and future use of the presented concept.

9.2 Related Work

Existing simulation and analysis software for power transmission and distribution systems is mainly aimed at industry professionals. Requirements for extensive knowledge in the field and access to data for a transmission network's components pose high barriers to entry for these programs.

PowerWorld Simulator [438], for instance, is a commercial product to interactively simulate large-scale power transmission systems in great detail. While there are educational and research licenses available to make it accessible for non-industry users, the project's source code and development are not open so its extensibility is rather limited.

Other projects like *OpenDSS* [439] and *GridLab-D* [440] are open source projects aimed at research and planning purposes. Both of these tools have extensive simulation and analysis capabilities and support a wide array of grid types and distribution elements, but they are not interactive simulators.

Additionally, all three of these systems require very detailed data about the elements of the transmission system to simulate, as they calculate all data of the network. While properties like reactive current and line frequency – both specific to AC power – are important in real systems, their meaning and impact is rather cryptic to the novice user. Beginners would have trouble finding, for example, the resistance and thermal properties of transmission lines, the efficiency of power plants and transformers, and many more data required to set up simulations in these existing systems.

9.3 PowerSurge Design

In this section, we detail the design concept of the PowerSurge software, including its user interface, gamification elements [416], and its underlying domain model.

9.3.1 Visualization of Simulated Units

To simplify the visual representation of the power transmission network, the system is based on the look and feel of a board game. Game pieces which can be placed on the playing field (a map of Germany) are *nodes* that represent power plants, consumers, distribution nodes and *transmission lines* which connect the nodes to form a network. The nodes are composed of a 3D model to represent the type of the node and a base plate on which additional data can be represented. The models are simplified but the optical characteristics of each node type allow for easy visual distinction by the user. The available node types are shown in Figure 9.1.

The diameters of the transmission lines symbolize their power capacity. The magnitude and direction of the power flow is visualized by the movement speed and direction of the stripes on the connections' surfaces. An example network of the major power plants and cities near Augsburg and Munich is illustrated in Figure 9.2.

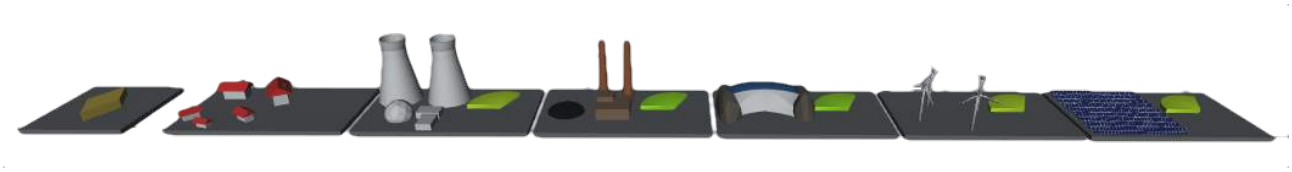


Figure 9.1: All node types available in our simulation. From left to right, these are: a distribution node, a consumer node, a nuclear power plant, a coal power plant, a hydro power plant, a wind park and a solar power plant.

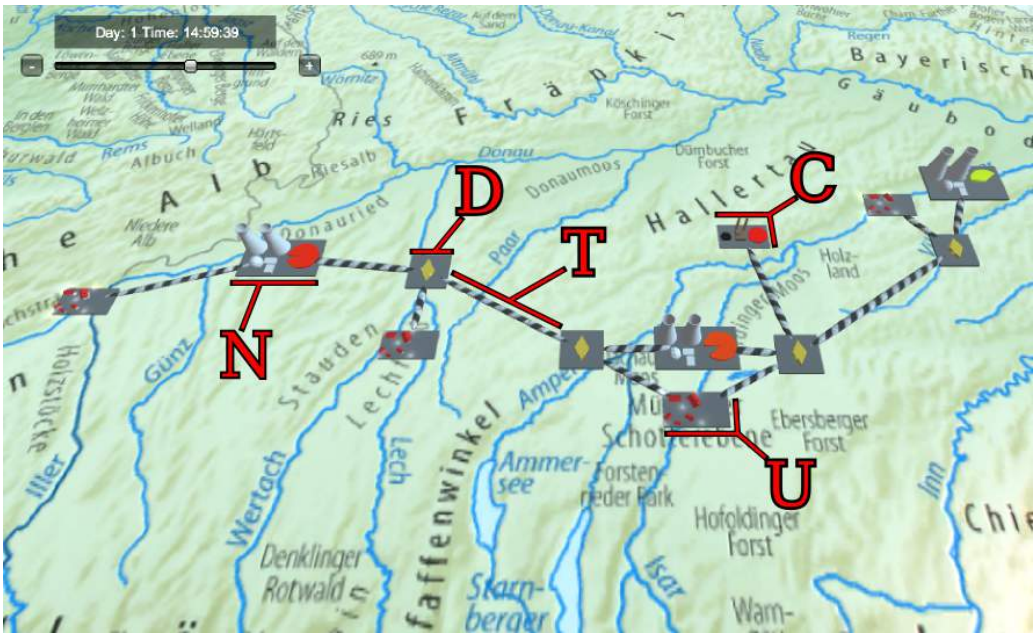


Figure 9.2: An example network. The visible game elements are: nuclear power plants (N), a coal plant (C), cities (U) and distribution nodes (D), all of which are connected by transmission lines (T).

We used various techniques to represent the current state of the network. (1) A pie chart on the generator nodes' base plates fills according to the current load. The indicator also changes its color in a gradient from green to red the closer the load gets to the maximum output. (2) The maximum power output of a node is represented by its size. When a node is dropped onto the field, its size is scaled up or down according to its maximum output relative to the network's overall power generation. Whenever the distribution of power generation among all power plants on the field changes, the relative sizes are updated to reflect these changes. The same technique is used for the transmission lines. The more power it can transport, the thicker the line gets. (3) To show the direction of flow and the amount of electrical power transported by a line, the black lines on each transmission line's surface move in the direction of the power flow. The movement speed depends on the amount of power transferred on this line relative to the maximum power flow of all lines on the network. This way the user can quickly determine

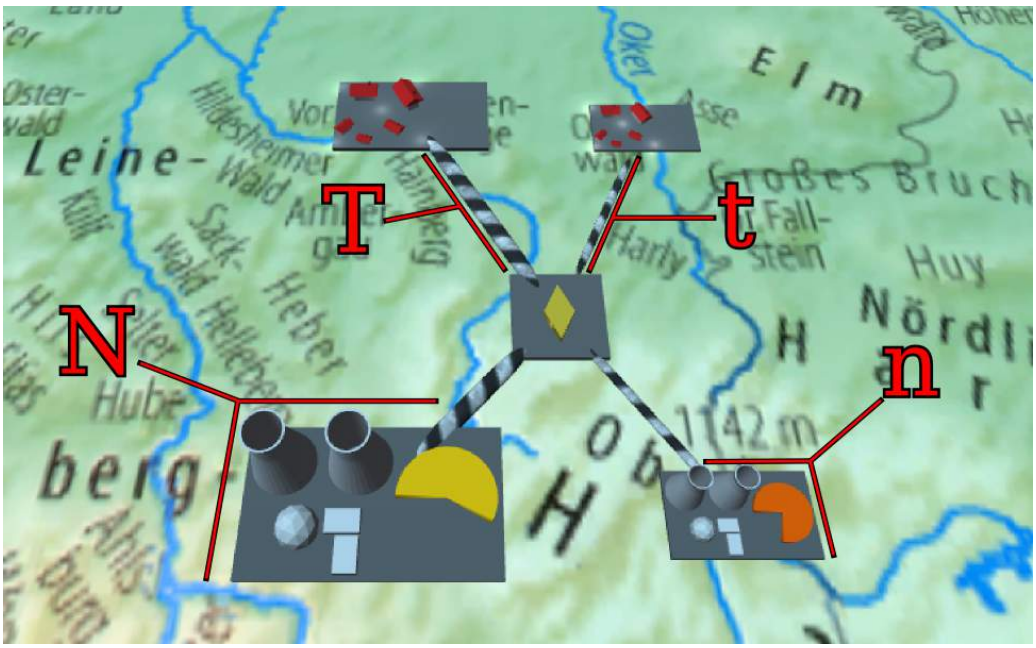


Figure 9.3: Nodes and transmission lines are scaled according to their current load or power output. The thicker transmission line (T) currently transports more power than the thinner line (t). Analogously, the larger power plant (N) generates more power than the smaller one (n)

which lines transport a lot of power and which lines don't. An example of these relative sizes is shown in Figure 9.3.

It is possible for a transmission line to be overloaded by a distinct percentage. While this should be avoided and is not a permanent solution for a stable network, it makes the network more resilient toward short bursts of power. In case a line is overloaded, the color of the white stripes on the line's surface turn red to alert the user to the problem.

Introspection of the various simulated units allows to access additional data such as the generator nodes' output (Figure 9.4), the consumer nodes' power intake, maximum load, and daily load patterns, or the power lines' transported power, their maximal capacity, and their possible overload. For an overview of the date of the whole network, the sidebar presents various information like the total produced and consumed power, the amount of power lost due to resistance, the percentage of fossil or renewable energy production and further more information.

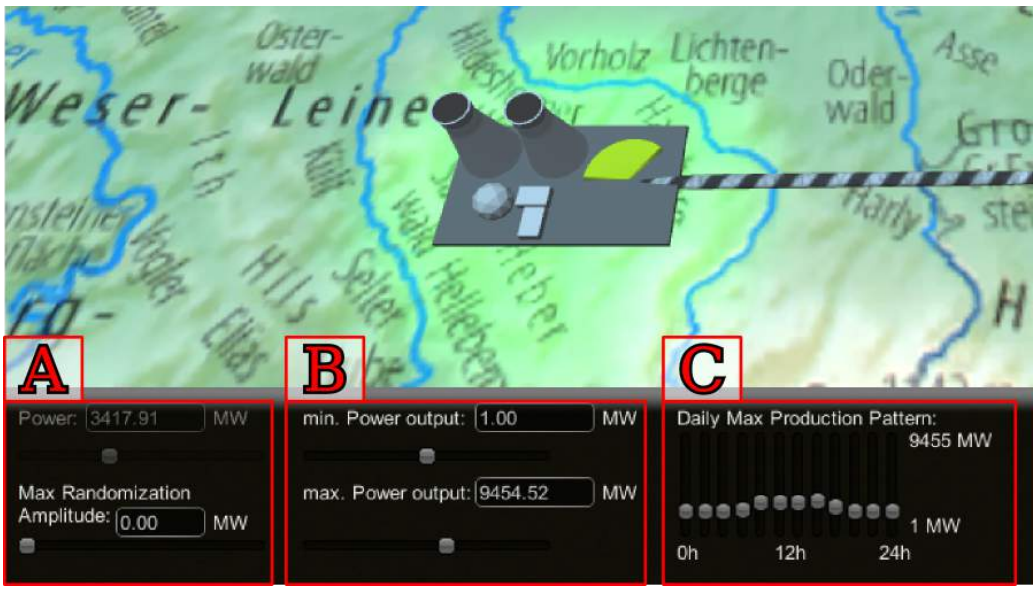


Figure 9.4: The information bar for a generator node shows its current power output and the maximum output randomization amplitude (A), the minimum and maximum power output (B), and the daily maximum production pattern (C).

9.3.2 User Interaction

In our simulation, it is possible to change the parameters of all nodes and transmission lines while the simulation is running, and the user will experience a real time adaptation of the network to the new settings. To add nodes to the system, the user can select the desired type of node from the sidebar and then drop it anywhere on the field, though it is not possible to drop nodes on top of each other. Connections can be drawn between individual nodes, selecting them in *connection* mode. Yellow and red backdrops of the targeted nodes indicate whether or not a connection can be established. By clicking on a node or transmission line, that game piece gets selected, which is indicated by a green highlight around the object. While a game piece is selected, the user may remove it from the game – as long as this action is not forbidden by the game scenario the user is challenged with. This case may also keep the user from inspecting an object and changing all its parameters as seen exemplarily in Figure 9.4.

9.3.3 Gamification

In order to encourage the user to explore the simulator the application offers three different stages of the game. We included a set of *tutorial levels* to introduce the user to the contents of the software, the meanings of the visualizations, the interaction mechanics, and the goals and

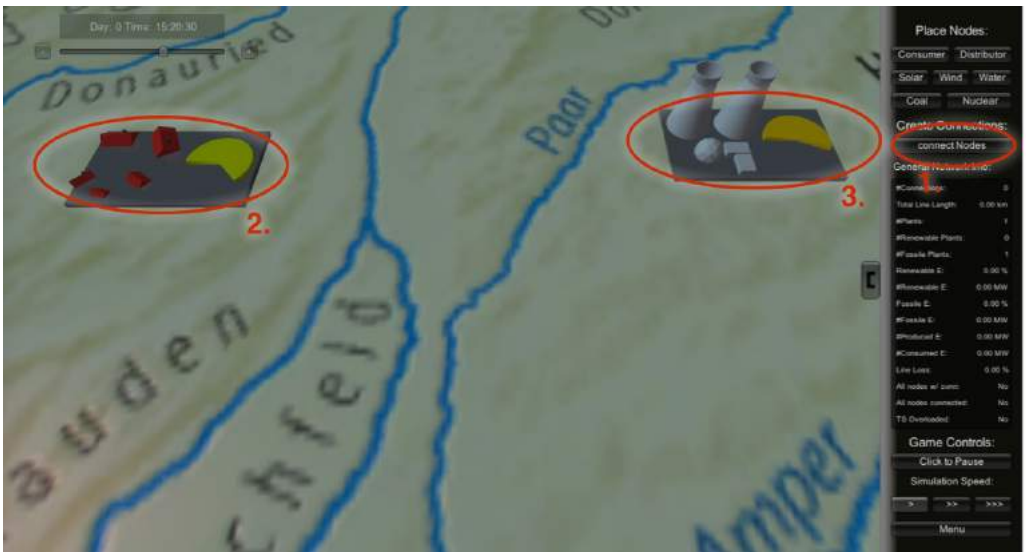


Figure 9.5: The first tutorial level to guide the user step-by-step through the interaction mechanics and to familiarize him with the simulated domain. In red, we hint at a sequence of steps that establishes the required connection to move on to the next level.

challenges of the simulated domain. The first tutorial level is exemplarily shown in Figure 9.5. As soon as the user is familiar with the basic interaction mechanics and the relationships of the simulated units, he can prove himself in scenarios of increasing difficulty. In the according *scenario mode*, the user has to overcome some predefined constraints in order to achieve certain goals. One scenario asks, for instance, to provide at least 20% of consumed power from renewable sources while using no more than 20 power plant and 30 transmission lines supplying a given set of consumers. The user is presented with a list of the subgoals for each scenario before it begins, and he can revisit this list at any time via the game menu. An example of this overview screen is shown in Figure 9.6. After fulfilling all the subgoals of a scenario, the user is notified and he may advance to the next one. If subgoals are no longer satisfiable, for instance when a time-limit is exceeded, the user fails in that scenario. An appropriate *game over* screen is shown and the user may restart this scenario or switch to a different one.

PowerSurge also includes a *discovery mode*, a scenario free of any constraints, where users can try out various model configurations on their own agenda. When the simulation is running in discovery mode, the user may construct arbitrary power transmission networks with no restrictions on size of the network, resources spent or network composition. Even the otherwise imposed fixed time sequence is now softened: here, the user may go back and forward in time as desired. All produced data, i.e. time series of all the graph's variables, can be logged on disk as portable comma-separated lists for further analysis. Through this interface for scientific

Figure 9.6: Exemplary display of the goals of a scenario. These goals are presented to the user before diving into the simulation.

evaluation, one could, for instance, measure network properties, such as topology, robustness (in terms of redundancy and minimum supply) etc.

9.3.4 Scripting Game Contents

Especially in an educational setting it is important to have the flexibility of defining specific problem scenarios. PowerSurge is designed to be extended accordingly. Goals and constraints of new scenarios can be scripted based on a variety of possible subgoals provided by the engine. An overview of these subgoals is shown in Figure 9.7. In the following, we describe some of these goals.

The first goal, i.e. *Max Time To Achieve Goals In Days*, sets a global time limit for all subgoals to be completed. Exceeding this limit, results in failing the scenario. The second one determines the minimum percentage of renewable energy produced in the Power Network. Additionally, a period can be set, how long the system has to supply that proportion continuously (using *AVGProduction Period Length*). Consider, for Instance, that at night there is less available renewable power available than on a bright sunny day, when solar radiation can also be harnessed. The goal, *Max Network Failure Time* denote the maximum limit of time a network failure is allowed to last. If *Network Failure Time Stacks* is set, that time will not be reset, if the system stabilizes again. A single-component graph is required, if the goal *Network Graph Must Be Complete* is set. This means that starting at any node, every other node on the playing field must be reachable. No disjoint network components are permitted. Each requirement is individually adjustable, being ignored if set to the default value (-1 or *disabled*). During

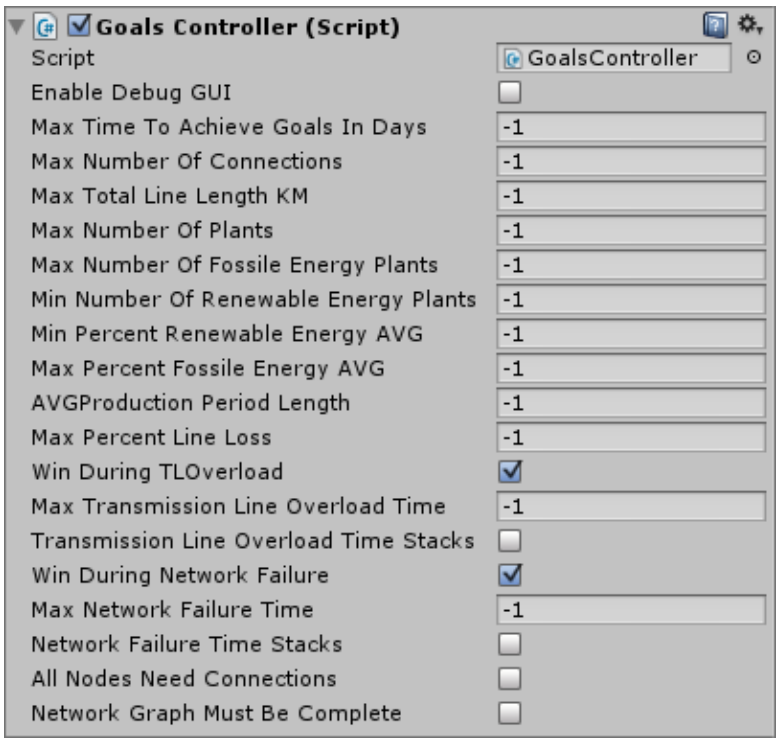


Figure 9.7: Scriptable goals for new scenarios (entries with the default parameter -1 are not considered for the evaluation).

the simulation, all of the subgoals are continuously tested to detect whether the overarching scenario goal is already met, or, in the worst case, cannot be reached any longer.

In PowerSurge, offering challenges to the user is synonymous with restricting his interaction possibilities, i.e. going from an all-flexible editor towards a concrete real-world problem perspective. Otherwise, the user could easily bypass the designed challenge by deleting lines or nodes, or by adjusting their production or throughput values. Therefore, we implemented a simple access rights management system for the simulated objects. As a result, user access can be individually adjusted for each object when setting up new scenarios. There are three access modes: (1) *Modify and delete* grants the user full rights to modify the object's properties or even remove it from the screen. (2) *Modify only* allows the user to modify all properties but not to remove the object itself. (3) *Sealed* means that the user can only view the placement's current properties to react to its behavior during play.

9.3.5 Domain Model

Formally, the power transmission network in our simulator is represented by an undirected graph. In this graph, the vertices or nodes are the consumers (e.g. cities), generators (power plants) and distributors (which neither produce nor consume power). All node types share one property, the power production/consumption P_{self}, which is positive/negative if the node generates/consumes power, or zero, if the node solely distributes power. In addition, the constant base consumption of a consumer node is captured by P_{base}. A load pattern defines the node's fluctuating power consumption over the course of a day. A generator node has both a minimum and a maximum power output value P_{min} and P_{max}, respectively. Its output is further modified by a function of time (consider day/night cycle), and a randomized fluctuation with the amplitude of $P_{randAmp}$. The nodes are connected by means of transmission lines. The resulting graph is irreflexive, i.e. no node can be connected to itself. A transmission line is characterized by the two nodes A and B to which it is connected, its length l, its maximum power load P_{max} and its maximum overload factor $f_{overload}$.

In addition to user-defined or scenario-dependent parameters, the simulation has to solve for specific variables, such as the actual power output of all generators and the power flow on all transmission lines. To derive these values, we set up a system of equality and inequality constraints modelling the behavior of the network. We then minimize an evaluation function within these constraints, using a boundary, linear equality and inequality constraints solver provided by the accessible and established open library ALGLIB [441, 442]. We were able to directly embed it into our development environment, Unity3D[1]. In the following paragraphs we present both the evaluation function and all constraints imposed on its optimization.

As the transmission lines are bidirectional, power on each line can flow in either direction. The power loss due to the line's electrical resistance depends on the amount of power flowing into the line. To properly apply the power conservation to the transmission line, we must therefore know in which direction the power flows. To accommodate this, we split each bidirectional power line into two unidirectional power paths in the context of the optimization. This means that a transmission line whose endpoints we call A and B has four optimization variables, power inflow and outflow for the path from A to B (P_{in}^{AB} and P_{out}^{AB}, respectively) and for the path

[1] `http://unity3d.com`

from B to A (P_{in}^{BA} and P_{out}^{BA}, respectively). The evaluation function h optimized by the solver is composed as follows:

$$h = \sum_{i \in lines} (L(i) + S(i) + O(i)) \tag{9.1}$$

where for a given transmission line i the power loss $L(i)$, the squared power values $S(i)$ and the overload $O(i)$ are defined as:

$$L(i) = \lambda^{l(i)} * P_{in}^{AB}(i) + \lambda^{l(i)} * P_{in}^{BA}(i) \tag{9.2}$$

$$S(i) = P_{in}^{AB}(i)^2 + P_{out}^{AB}(i)^2 + P_{in}^{BA}(i)^2 + P_{out}^{AB}(i)^2 \tag{9.3}$$

$$O(i) = \begin{cases} E(i)^2 & \text{if } E(i) > P_{max}(i) \\ 0 & \text{otherwise} \end{cases} \tag{9.4}$$

$$E(i) = \max\left(|P_{in}^{AB} + P_{out}^{BA}|, |P_{in}^{BA} + P_{out}^{AB}|\right) \tag{9.5}$$

where λ is the power retention factor per km of the transmission lines, $l(i)$ is the length of transmission line i, $P_{in}^{AB}(i)$, $P_{out}^{AB}(i)$, $P_{in}^{BA}(i)$ and $P_{out}^{BA}(i)$ are the power inflow and outflow of the two paths of transmission line i as explained above, and $P_{max}(i)$ is the maximum load of transmission line i. Power loss on the transmission lines (9.2) is the main criterion we want to minimize, so its inclusion is obvious. It should be noted here that the power loss is technically

$$P_{loss} = R \cdot I^2 = R \cdot \left(\frac{P_{flow}}{U}\right)^2 \sim P_{flow}^2$$

where R is the line's electrical resistance, U is the line voltage, I is the current and $P_{flow} = U \cdot I$ is the total power flowing on the line. While the evaluation function could handle this quadratic function, the constraints for the optimizer must be linear in all variables. We therefore opted to use the linear relation $P_{loss} = R' \cdot P_{flow}$ (where $R' = \lambda^l$ is the power retention factor which acts as a stand-in for the line resistance) in both the constraints and the evaluation function in order to keep our system consistent. Including the sum of the squared endpoint power values (9.3) in the evaluation function minimizes the total power flow on the network. On the one hand, this serves to prevent power flow over detours – the more line endpoints need to be traversed by power flowing from generators to consumers, the more it factors into this summand. On the other hand, this prevents power from flowing in both directions at once on any transmission line, which would incur more power loss while reducing the net power

transported over the respective line. Lastly, the overload summand (9.4) ensures that line overloading, while allowed, is discouraged. To achieve this, the overload summand is zero if the line is not overloaded. Once the line load enters the overload interval, this summand contributes the squared effective power flow of the line.

When searching for the equilibrium of a given power transmission network, the evaluation function discussed above is minimized with a set of constraints. This set is composed of two constraints for each node and eight constraints for each transmission line present in the network. A node n of any type must meet the power balance constraint:

$$P_{self}(n) + \sum_{i \in cons(n)} (P_{in}(n,i) + P_{out}(n,i)) = 0 \tag{9.6}$$

where $cons(n)$ is the set of all transmission lines connected to node n, $P_{in}(n,i) \leq 0$ is the power flowing from n into the line i, and $P_{out}(n,i) \geq 0$ the power flowing from i into n. This equality constraint (9.6) states that the total amount of power flowing into a node n must equal the total amount of power flowing out of the node. In other words, taking the node's own power generation or consumption P_{self} into account, no power may "magically" appear or disappear on the node. As previously mentioned, a distribution node n_d neither generates nor consumes power, so its local power is constrained to 0:

$$P_{self}(n_d) = 0 \tag{9.7}$$

The local power of a consumer node n_c must exactly match the node's current consumption:

$$P_{self}(n_c) = P_{consume}(n_c, frac(t)) \leq 0 \tag{9.8}$$

where the power consumption $P_{consume}(n_c, t)$ at simulation time t (in days) is determined by the load pattern of the consumer node over the course of a day. As the simulation time t is given in days, we extract its fractional part $frac(t)$ as the time of day. On a generator node n_g, the local power is constrained by the power output bounds:

$$0 \leq P_{min}(n_g) \leq P_{self}(n_g) \leq P_{max}(n_g, t) \tag{9.9}$$

where the simulation time dependent maximum power output $P_{max}(n_g, t)$ is determined as:

$$P_{max}(n_g, t) = \; P_{pattern}(n_g, frac(t)) + \; uniform(-P_{randAmp}(n_g), P_{randAmp}(n_g))$$

The constant parameter $P_{min}(n_g)$ is the plant's minimum power output boundary. The plant's maximum power output is determined by a pattern $P_{pattern}(n_g,\, frac(t))$ analogously to the load pattern of consumer nodes. For renewable energy generation like wind and solar power, which are heavily subjected to natural fluctuations, an additional summand samples a uniformly random value within the given randomization amplitude $P_{randAmp}$. This fluctuation can be reduced or completely disabled for other plants, e.g. nuclear power plants, by setting $P_{randAmp} = 0$. For each transmission line i, these eight constraints apply:

$$\lambda^{l(i)} \cdot P_{in}^{AB}(i) + P_{out}^{AB}(i) = 0 \qquad \lambda^{l(i)} \cdot P_{in}^{BA}(i) + P_{out}^{BA}(i) = 0 \tag{9.10}$$

$$P_{in}^{AB}(i) \leq 0 \qquad P_{in}^{BA}(i) \leq 0 \qquad P_{out}^{AB}(i) \geq 0 \qquad P_{out}^{BA}(i) \geq 0 \tag{9.11}$$

$$|P_{in}^{AB}(i)| \leq f_{overload}(i) \cdot P_{max}(i) \qquad |P_{in}^{BA}(i)| \leq f_{overload}(i) \cdot P_{max}(i) \tag{9.12}$$

The equality constraints (9.10) represent the power balance on the transmission line i. The amount of inflowing power $P_{in}^{AB}(i)$ or $P_{in}^{BA}(i)$, scaled by the line's power retention factor $\lambda^{l(i)}$, must equal the amount of outflowing power $P_{out}^{AB}(i)$ or $P_{out}^{BA}(i)$, respectively. The boundary constraints (9.11) state that the variables for power flowing into the line, $P_{in}^{AB}(i)$ and $P_{in}^{BA}(i)$, must have a negative sign, while those for power flowing out of the transmission line, $P_{out}^{AB}(i)$ and $P_{out}^{BA}(i)$, must have a positive one. Notice that it suffices to include only one of the inequality pairs (9.11) in the actual optimization. Together with the power conservation constraint (9.10), either of these boundary pairs implies the other one. Finally, the inequality constraints (9.12) ensures that power flow on no transmission line exceeds that line's effective maximum load, which is the product of the line's regular maximum load $P_{max}(i)$ and its maximum overload factor $f_{overload}(i)$.

9.4 Results & Future Work

We designed a system that allows the users to interactively learn about and explore the complexities inherent in power transmission systems. Several basic tutorials cover the basic relationships of producer and consumer nodes, introduce the intricacies of patching overloaded networks, and to minimize the utilization of resources such as the overall length of the power lines.

In a competition on interactive simulations, we presented PowerSurge to about 30 people, most of them students. They voted PowerSurge to be the best out of seven projects, including interactive simulations in domains as far apart as biology and traffic systems. Criteria in the competition comprised the complexity of the scientific model, the usability, and the visual appeal.

At this point, important aspects such as local energy storage systems are still missing, however, an understanding of the complex interplay between producers and consumers can already be gained. Apart from improving the functionalities of the simulator, we deem the following aspects as especially beneficial extensions: A highscore system could lead to a better grasp on the user performance and build social ties between the users, which is an important factor of motivation. Along these lines, multi-player modes could promote the collaborative (re-)design of power grids at moderate scales or increase the fun through one-on-one or team competitions. An extension of the time system could allow the user to go back in time and review the changes that were made to the system. Also it should be possible to create different branches of the timeline to tackle some of the problems with different ideas. New scenarios should allow for the definition of more constraints, for example the number of each type of node that can be planted. In addition to numerous other user-centered and technical improvements, we are planning to test our software as part of an educational curriculum.

Chapter 10

OCbotics: An Organic Computing Approach to Collaborative Robotic Swarms

In this paper we present an approach to designing swarms of autonomous, adaptive robots. An observer/controller framework that has been developed as part of the Organic Computing initiative provides the architectural foundation for the individuals' adaptivity. Relying on an extended Learning Classifier System (XCS) in combination with adequate simulation techniques, it empowers the individuals to improve their collaborative performance and to adapt to changing goals and changing conditions. We elaborate on the conceptual details, and we provide first results addressing different aspects of our multi-layered approach. Not only for the sake of generalisability, but also because of its enormous transformative potential, we stage our research design in the domain of quad-copter swarms that organise to collaboratively fulfil spatial tasks such as maintenance of building facades. Our elaborations detail the architectural concept, provide examples of individual self-optimisation as well as of the optimisation of collaborative efforts, and we show how the user can control the swarm at multiple levels of abstraction. We conclude with a summary of our approach and an outlook on possible future steps.

Sebastian von Mammen, Sven Tomforde, Jörg Hähner,
Patrick Lehner, Lukas Förschner, Andreas Hiemer, Mirela Nicola, Patrick Blickling:
OCbotics: An Organic Computing Approach to Collaborative Robotic Swarms. In: *2014 IEEE Symposium on Swarm Intelligence (SIS)*, IEEE, 2014, pp. 1–8.

10.1 Introduction

In order to benefit from an ever more complex technical environment, its behavioural autonomy needs to increase appropriately as well. Only then, it may serve its users without requiring overwhelming amounts of attention. At the same time, a technical system is expected to offer appropriate access for controlling its individual components as well as its global goals. The control of robotic swarms lends itself well to elucidate this challenge: Ideally, the user would communicate his goals to the swarm as a whole, without the need of micromanaging the individuals' every parameter and interaction. For instance, the user might navigate a flock of flight-enabled robotic units towards a building and make them work on facade maintenance, e.g. scrapping off paint, cleaning windows, or trimming greenery. For this to work, a line of command has to be established that links several levels of the system's architecture—the user needs to communicate target and task to the swarm and the swarm individuals communicate to coordinate their efforts. In addition, each swarm individual needs to learn how it can contribute to the newly posed, global goals, and how it can maximise its contribution.

The field of Organic Computing (OC) aims at translating well-evolved principles of biological systems to engineering complex system design [115]. It provides the theoretical underpinnings to quantitatively capture system attributes such as their autonomy and robustness, or processes of emergence based on measures of entropy. It also promotes complex system design by means of a universal, observer/controller-based architecture for adaptive, self-organising behaviour. With respect to robotics, OC research initially focussed on failure tolerant and robust hardware architectures, mainly applied to multi-legged walking machines. The most prominent example is the *Organic Robot Control Architecture* (ORCA) [443, 444]. In ORCA, two kinds of behavioural modules are discerned. *Basic Control Units* (BCUs) implement the core behaviour of the robot, rendering it fully functional with respect to the range of possible tasks. In addition, *Organic Control Units* (OCUs) observe and modify the BCUs' configuration during runtime. The separation between a system's basic and its extended functionality has proven itself numerous times—the sympathetic and the parasympathetic division of the human autonomous nervous system may serve as a famous biological example.

Similarly to ORCA, we follow an OC approach to self-organising robotic systems. In our approach, each agent in a robotic swarm implements a multi-layered observer/controller (O/C)

architecture that allows for local, and in unison, also in global optimisation of the swarm agents' behaviours. The user interface is explicitly included as one layer which accepts modifications of the swarm's and the individual agents' goals. We present the details of the multi-layered O/C architecture of a single swarm robot individual and we explain how it works in organising ensembles (Section 10.2). In Section 10.3, we give an example of the reactive, self-regulatory capacity of the architecture. Section 10.4 highlights the longer-term evolution of collaborative behaviour, and Section 10.5 demonstrates the workings of the user interfacing layer of the architecture. We provide links to related works in the respective sections, and we conclude with a brief summary and an outlook on future work.

10.2 The OCbotics Approach

As mentioned in the introduction, our approach relies on an architectural setup similar to ORCA. Therefore, we first reinforce the link between our approach and ORCA and related works. Next, we build on these analogues to preceding works to detail our approach—from the perspective of a generic architecture as well as of its concrete implementation.

10.2.1 From Single Adaptive Units to Teams

In ORCA, the Organic Control Units change the system under observation and control (SuOC) based on periodically issued *health signals*, i.e. messages from the Basic Control Units indicating their functional working state. In contrast, our approach observes all kinds of available data about the SuOC. An according *observation model* specifies exactly, which input data, configuration parameters, or internally computed results of the SuOC are passed on to the observer/controller layer. ORCA's restrictive policy of data retrieval matches its fairly conservative array of options for changing the system. Few choices, however, drastically limit the system's configuration space and thus promote ORCA's primary design goals of (a) unearthing an optimal learning guidelines for adaptation ("the law of adaptation"), and of (b) protecting acquired knowledge against corruption and maintaining its validity and consistency [445].

The ORCA approach is further limited to single, isolated robots—information exchange with other robots or collaborative efforts among robotic teams were not envisaged in the original

architecture. Yet, it has been shown that Observer/Controller-driven robots can increase their learning speed imitating each other [446]. Local communication between robots allows for establishing real teams that collaboratively perform tasks such as the exploration of unknown terrain, and that assign each other subtasks in a fair manner—decentralised, without the need for global control [447, 448].

10.2.2 O/C Architecture

As suggested above, the OCbotics approach is founded in a multi-level observer/controller architecture. An according diagram is presented in Figure 10.1. It shows four interwoven architectural levels. Level 0 denotes the system under observation and control, the base of the architecture located at the bottom of the figure. Immediately above, level 1 retrieves and evaluates data about the SuOC's performance. Based on this data, it changes the SuOC's configuration in order to optimise its performance, to adapt it to varying conditions and needs. In particular, the SuOC's parameters/behaviours are optimised that may result in both good and bad performance values with respect to a predefined goal (introduced by level 3). As a consequence, the best possible configuration set, or behaviour, known to level 1 is exhibited by level 0 at any given situation. True innovation is realised by level 2, one step above in the multi-level architecture. Here, completely new behavioural options are generated, simulated and optimised in a sand-boxed simulation environment. Only if the new model specifications satisfy all safety constraints considered as part of the simulation process, they are eventually fed into level 1.

10.2.3 Modified XCS

Several studies in Organic Computing have emphasised the adequacy of Learning Classifier Systems (LCS) as a comprehensive framework to support adaptive observer/controller architectures, see for example [449, 450]. Already the first LCS presented in 1975 combined basic rule-based reactive behaviour with an evolutionary component to evolve and improve the rule base [451]. With the introduction of accuracy-based reinforcement of classifiers, LCS research reached an important milestone in 1995 [452] (for an overview of LCS research and ongoing research topics see, for instance, [453]). In the context of safety-critical Organic Computing

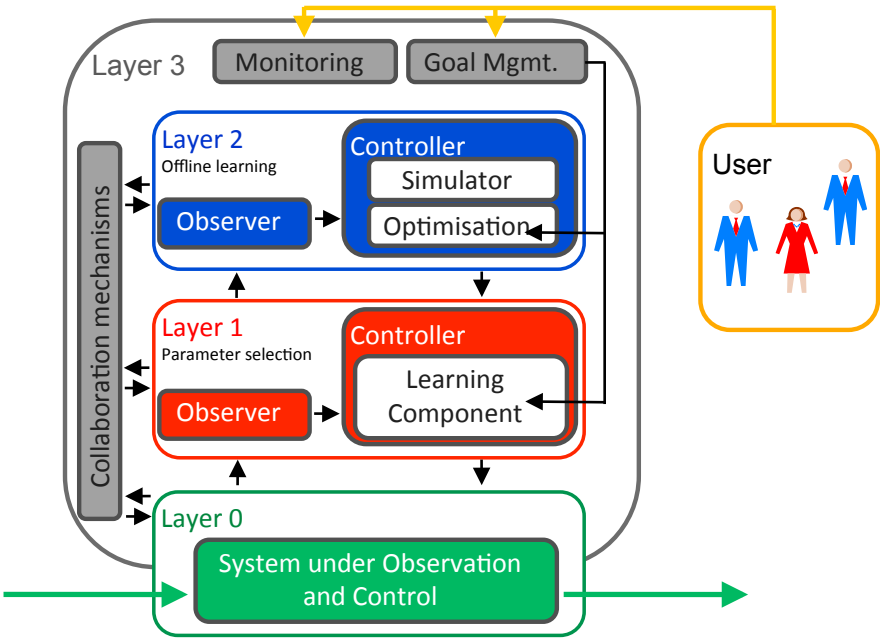


Figure 10.1: Multi-level Observer/Controller Architecture. The System under Observation and Control (SuOC) at the bottom is observed and adjusted by the O/C layers above. Their responsibilities are (in this order): reinforcement of existing behaviour, innovating new behaviour, and interfacing local behaviour with (a) user-set targets and (b) cooperative units' goals.

applications, the latter extension to LCS, also referred to as XCS, was further modified to suit the multi-layer O/C architecture outlined above. In particular, three modifications were implemented: (1) The use of continuous value ranges as promoted in [454], (2) the generalisation of the closest fitting existing rule in layer 1 instead of the generation of a new rule, in case that a given situation is not covered by the existing rule set ("widening" covering mechanism), and (3) "sandboxed" offline learning in layer 2 to ensure safety and maturity of new rules/behaviours. In addition, one can track the impact of those triggered rules effecting changes identical to the newly generated rule and, thereby, building up trust in new rules before they are considered by themselves.

In the remainder of this paper, we show preliminary examples of the OCbotics approach each of which works at a different level of the presented architecture. In particular, we show an example of reactive behaviour of a particular system under observation and control (layer 0) and we elaborate on its integration with layer 1 (Section 10.3). An instance of evolved behaviour (layer 2) in a collaborative swarm robotics setting is presented in Section 10.4. Its communication across a swarm of agents as well as the interface mechanism with the user of the system, i.e. layer 3, is explained in Section 10.5.

10.3 Self-organised Aerial Robotic Construction

Tensile structures play an important role in post-modern architecture [455] and they promise to become increasingly important still considering their unique versatility and flexibility in combination with advances in technologies in built material and construction methods [456]. They have also been subject to Aerial Robotic Construction (ARC) research [457, 458] due to their light mass, load-carrying ability, and their ability of connecting large distances. Quadrotors have been identified as vehicles apt for aerial manipulation mainly due to their robust flight behaviour and their hovering capability [459]. In [458], prototypic building primitives such as single and multi-round turn hitches, knob and elbow nots as well the trajectories resulting from their concatenation have been discussed. Different from pre-calculating trajectories, we have been working on a self-organising approach to building tensile structures. We detail our approach below, followed by elaborations on its OCbotic-specific features.

10.3.1 Stigmergic Web-weaving

Typically, a spider weaves its web by itself [460, 461]. Complex web constructions, however, may require collaborative entanglement and tightening of ropes. This can, for instance, be achieved by synchronised flight through pre-calculated control points to cross the ARC quadrotors' trajectories. Alternatively, the swarm individuals may coordinate themselves relying on local stimuli, like social insects do [229, 230]. In this section, we present a first such locally motivated ARC experiment[1]. Currently, it involves only one quad-rotor that tightens a rope around a tent pole's four suspension lines, see Figure 10.2.

Technical Setup

For our lab-experiments, we currently employ the AR.Drone Parrot 2.0 quad-rotor system. It is connected to a standard PC via WLAN. The PC retrieves the sensory data of the quadrotor and issues the according navigational instructions. We make use of the quad-rotor's VGA camera that has a 90° field of vision, built-in image-processing capabilities such as marker recognition, and the estimates of its ultrasonic distance sensor. As this sensor and a downward

[1] Please find an accompanying video at `http://youtu.be/tZNeL-n1dDE`.

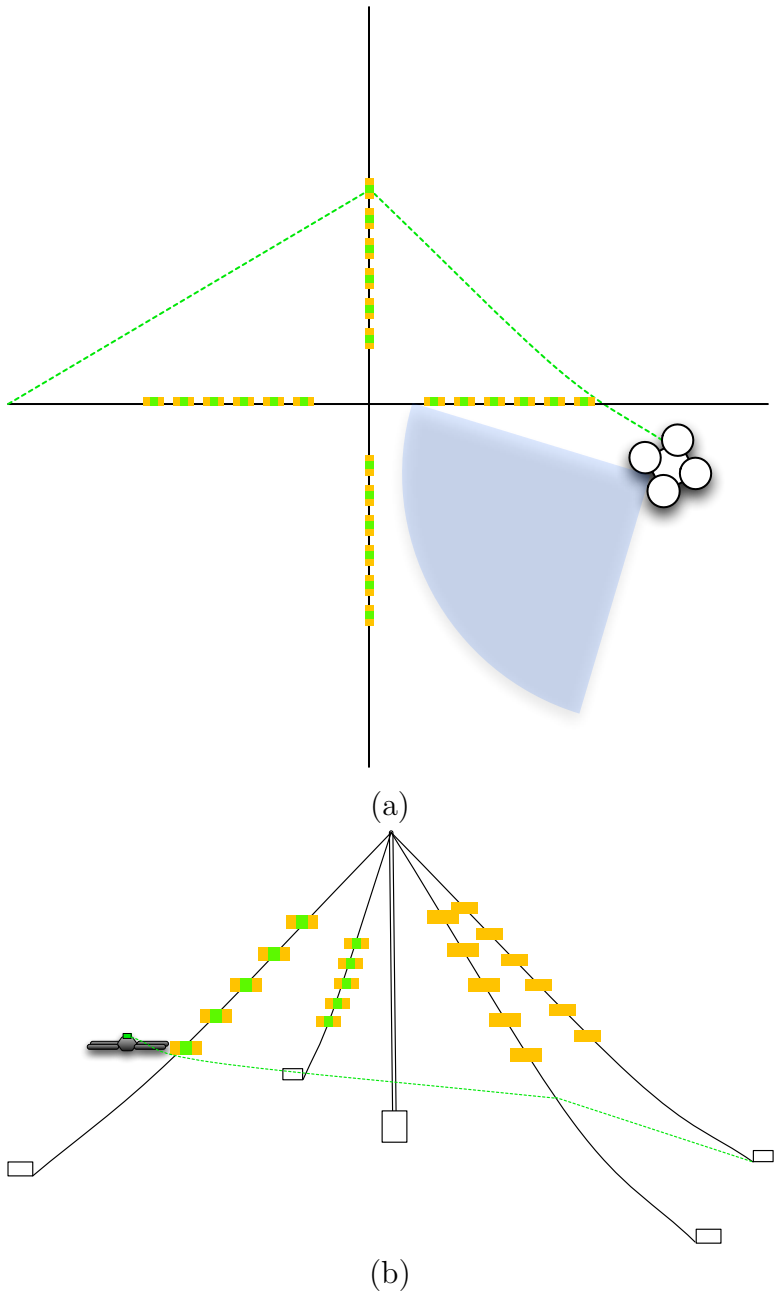


Figure 10.2: (a) The quad-rotor hovers clock-wise around a pole that is suspended by four lines. It tightens a rope (green, dashed) along the suspension lines. (b) A schematic side-view extracted from a photograph, highlighting the orientation of the markers pinned to the suspension lines.

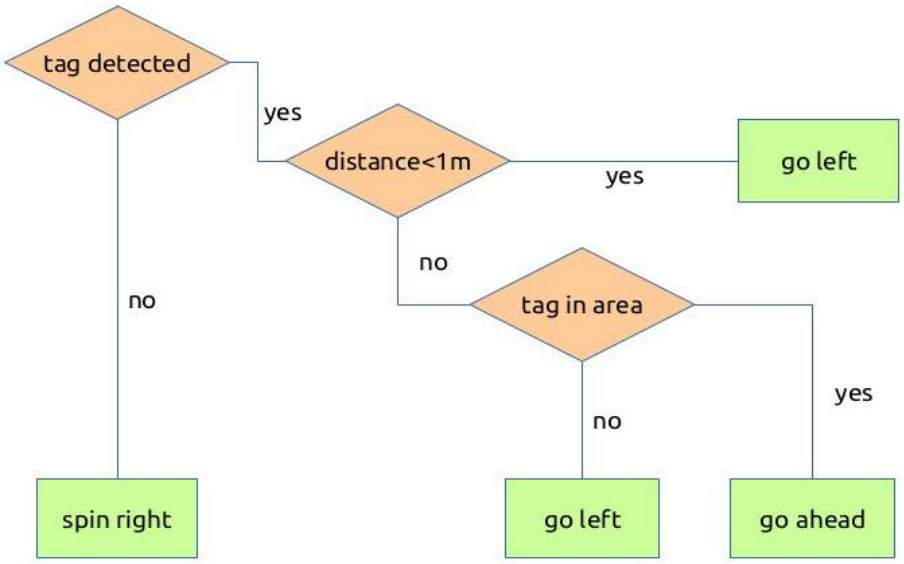


Figure 10.3: Weaving Behaviour of a self-organising ARC Quad-rotor. It circles clock-wise around a pole, tightening its thread around suspension lines tagged with directionally oriented markers.

directed camera are used by the quad-rotor to stabilise its flight, we attached a coil at the top of the vehicle and unwind the cord through an eye at its back. We interface with the quad-rotor relying on Nodecoper.js and the node-ar-drone module [462].

Behavioural Definition

The quad-rotor behaves only based on locally available sensory information. In particular, it implements the reactive behaviour schematically summarised in Figure 10.3: After taking off, it searches for a orange-green-orange marker, which is one of the designs that the vehicle is programmed to recognise automatically. It keeps spinning right until it eventually finds one. If the distance to the marker is less than a certain threshold (1m worked quite well), it drifts left. As a consequence, the detected marker moves outside of its field of view. At this point, the quad-rotor has surpassed the previous marker and looks for the next one, which is attached to the next suspension line (also consider Figure 10.2). The distance to the next marker along the circumference of the pole is greater than the given threshold. The quad-rotor can go straight ahead, if the tag is within the right-hand side of its view (this condition is labelled ‘tag in area’ in Figure 10.3). Otherwise, it needs to shift a bit to the left.

10.3.2 Adapting Reactive Behaviour

Programmatically, the quad-rotor's behaviour (Figure 10.3) is represented as a set of simple *if-then* rules. As such, they can be easily subjected to standard LCS implementations and its extensions such as XCS (Section 10.2.3). Hereby, those rules with the best prediction accuracy are reinforced to gain the greatest fitness over time, yielding the best possible behaviour. In this way, the quad-rotor of the ARC example would learn to query the proper sensors at the right times to react in the best possible way, if the behavioural rule set was enriched with according alternatives. At the interface of level 0 (the SuOC) and level 1 (the reinforcement learner), the measure of success can typically be calculated based on locally available information such as the distance flown or the number of recognised tags. For good learning results, the parametrisation of the behaviour should be realised at a rather high level, focusing on the selection of queries and operations and only cover small ranges of variability. Potential benefits of level 1 learning would not only be optimisation of one particular learning pattern but also behavioural rules that adapt to hardware particularities such as deviating sensory intake or imbalanced motor control.

10.4 Collaborative Aerial Robotic Maintenance

At level 2 of the multi-layer O/C architecture, behaviours can be created by means of generative model building approaches such as evolutionary algorithms and be optimised for deployment by means of simulations. As a first OCbotics prototype of offline level 2 generation and optimisation, we have evolved quad-rotor behaviour for collaborative surface maintenance. In this section, we introduce the challenge of optimising collaborative surface maintenance. We detail the technical setup we relied on for both simulation and optimisation and we describe the behavioural options of each swarm individual. Afterwards, we draw a very rough picture of the evolutionary experiments that we have run, and we discuss the interactions between layer 2 and 3 for propagating successfully bred behaviours that require synchronisation between the individuals in an OCbotics swarm.

10.4.1 Collaborative Facade Maintenance

Consider the facades of large office buildings as examples of vertical surfaces: They are subject to cleaning [463, 464], trimming greenery [465], and other maintenance tasks. As in the previous example, these tasks might benefit more from collaborative efforts than only in terms of efficiency. For instance, fast growing greenery might require one machine to bend, the other one to cut a branch. Equally, during cleaning, several hovering robots might have to join to build up sufficient pressure to remove persistent dirt. Of course, the respective operations might also be split into several procedures performed by individually optimised machines. In this example, however, we only consider the most modest objective, namely collaborative efficiency.

Technical Setup

The technical setup of our level 2 experiment comprises (a) a simulation environment to calculate aviation and robotic mechanics, and (b) a machine learning environment with a generative model component and an optimisation component. Figure 10.4 depicts the software modules that we have used in order to simulate collaborative quad-copter swarms. The Robot Operating System (ROS) acts as a hub for these modules. It provides a high-level software interface for programming and communicating with different kinds of robots [466]. Gazebo is a simulation engine that natively integrates with ROS, offering 3D rendering, robot-specific functionality and physics calculations [467]. Thanks to a ROS driver for the AR.Drone Parrot quad-rotor [468], and thanks to a Gazebo plugin that simulates the quad-rotor's behaviour based on the very same ROS-based instruction set [469], any of the generated behaviours immediately work in-silico and in-vivo. For the generation of novel behaviours as well as for their evolution, we decided to use the Evolving Objects framework (EO) [470]. EO is an open framework for evolutionary computation featuring an extensible, object-oriented architecture, and turnkey implementations of genetic algorithms, particle swarm optimisation, and genetic programming.

Behavioural Definition

Our approach to collaborative facade maintenance is inspired by nest construction of social insects [229]. Each individual works on a small part of the construction proportional in size to

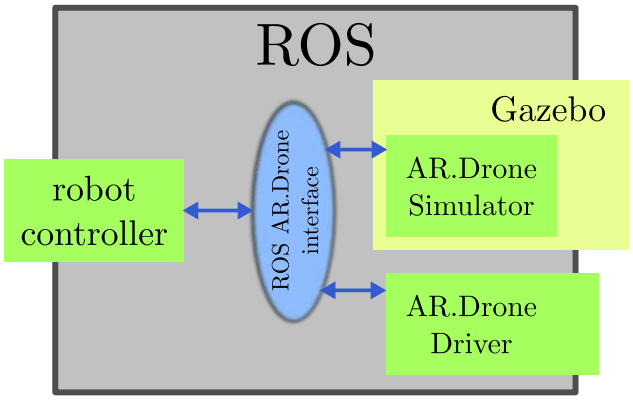


Figure 10.4: Interwoven Simulation Modules. The Robot Operating System integrates various software components to simulate quad-rotor swarms. In particular, we control the robot using a common ROS-based instruction set that is understood by both the Gazebo simulation software and its quad-rotor simulation plugin as well as a ROS driver that steers the actual quad-rotor hardware.

the insects' physique. Accordingly, each simulated quad-rotor divides the target surface in a grid, each cell measuring 2 by 2 metres, its field of view covering six cells, two rows of three (Figure 10.5). This partitioning scheme is a result of the size of the quad-rotor itself and its perceived area from a vantage point close to the surface. Without loss of generality, a dirtiness value is assigned to each cell that indicates whether it needs to be worked on or not. The quad-rotor's internal state, i.e. its remaining battery life, as well as the configuration of dirty and clean cells that reveals itself in front of it trigger specific actions. The quad-rotor may return to the base station to recharge. It may fly to one of the cells in its field of view and clean it. Alternatively, it may move to one of the four neighbouring vantage points to inspect the respective neighbouring batches of cells.

10.4.2 Evolving Collaborative Behaviour

Figure 10.6 captures the behavioural options of a quad-rotor in the context of facade maintenance. Any activity is initiated by the decision-making component, subsequent events guide the quad-rotor back into the decision-making process. Again, the behaviours can easily be written as *if-then* rules which ensures the coherence and simplicity of interfacing across the layers of the O/C architecture. Notice that in this model, quad-rotors cannot stop working. Instead, the whole simulation is terminated after a given amount of time. During this period of time, the decision-making component determines the success of the simulated swarm. We generate

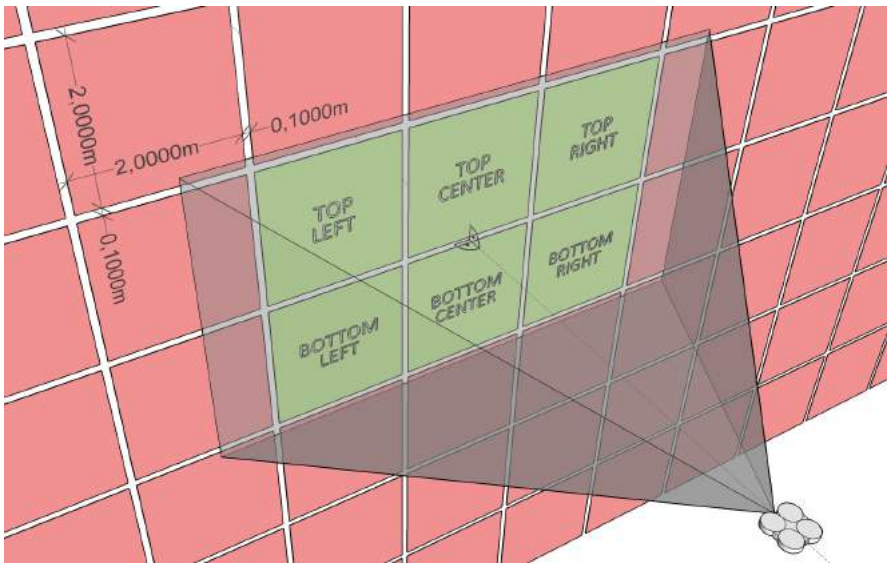


Figure 10.5: Cell Grid for Surface Maintenance. The quad-rotor divides the building facade in a grid of cells. The individual cells represent the immediate target areas to work on. Their states of cleanliness also provide the local cues for decision-making, i.e. for approaching an individual cell or to moving to another vantage point.

an according program tree using Genetic Programming [471]. In this paper, we refrain from presenting the evolutionary approach in all detail, but we want to convey its basic mechanisms and how it ties into the OCbotics approach. The generated decision program may conditionally deploy the operations outlined in Figure 10.6, and introduce references or primitive values as their parameters. The resultant behaviour trees are considered the individuals in an evolutionary optimisation cycle, and thus their fitnesses (penalty, a.o., for remaining amount of dirt) are calculated in according simulation runs, and they serve as an important criterion for selecting ancestors for subsequent generations of individuals (deterministic tournaments).

We ran experiments featuring two or four quad-rotors, or "aerial robotic units" (ARUs), working in parallel for 900 to 1500 simulated seconds. Their individual base stations were distributed randomly in rectangular area sharing two sides with the target surfaces as seen in Figure 10.7. Population sizes varied from 30 to 100 individuals, the generational cycle was repeated between 10 to 50 times—depending on the work load of an individual simulation which was mainly determined by the number of interacting agents and the simulated time. One of the best individuals in an experiment that started from a set of previously evolved specimen worked as follows: Having arrived at a random cell of the target surface, work through single rows of vantage points from right to left. If the border of the target surface is reached, return to the base station and approach the target area again. It turns out that this behaviour

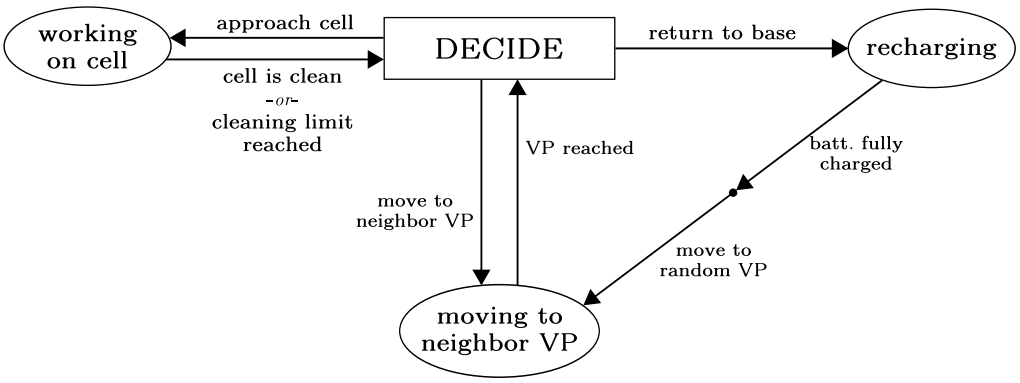


Figure 10.6: Options of Activity of a Facade Maintaining Quad-rotor Agent. Any activity—working on one of the cells ahead, flying to the base station to recharge, or moving to a neighbouring vantage point—is initiated by the decision component which considers the agent's battery state and the surface configuration in its field of view.

proved significantly faster than two decision programs we manually designed before running the evolutionary process: One of them stochastically selecting dirty cells and considering the remainder of the battery before taking action (low batteries are also penalised by the fitness calculation), the other one letting the quad-rotor follow the dirt gradient exhibited in the perceived 3 by 2 cell matrix.

10.4.3 Sandboxed by Layer 2, Letterboxed by Level 3

Level 2 is capable of generating and evolving collaborative behaviour such as the one described above. Initially, the novel behaviour does not have any impact on the system under observation and control. One may say the innovation process is encapsulated in a sandbox and runs completely separated process, offline. At the same time, collaborative behaviour needs to be communicated, if it is required to be performed by all individuals of a swarm in order to function in a coordinated way. The observation of Layer 2 by Layer 3 has to detect such impending necessary changes and broadcast it to all the other members of the swarm. Similar to an auction in multi-agent systems [472], the best broadcast solution, i.e. decision program and fitness value, would be implemented. As an extension, any population-based simulation and optimisation approaches could be distributed among the OCbotics individuals and their evolution be concerted across the whole swarm (for distributed population-based optimisation see, for instance, [473] and [474]). Especially in situations with imbalanced computational loads across the swarm, following a smart distributed optimisation strategy could yield an important

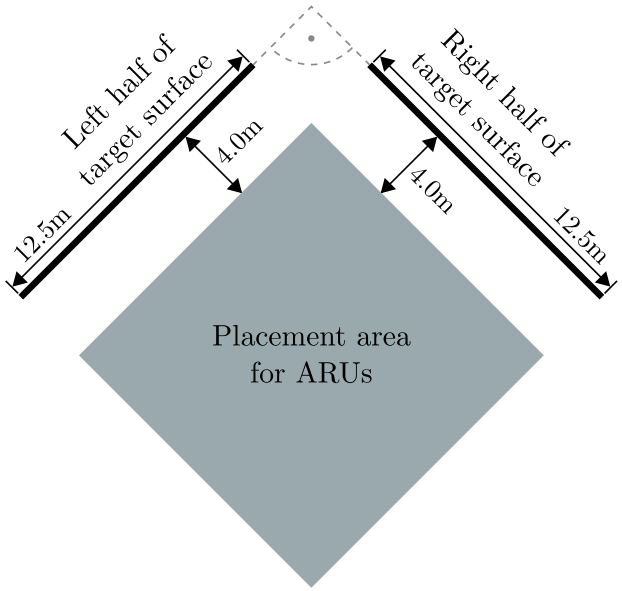


Figure 10.7: Simulated Environment for Collaborative Surface Maintenance Evaluation. Two flat surfaces are presented to the quad-rotor swarms as target area which needs to be cleaned. The base stations of the swarm individuals are randomly placed in the rectangular area between the surfaces.

advantage.

10.5 User-guided System Behaviour

In our last example, we demonstrate an early prototype of the user interfacing component of layer 3 of the multi-layered O/C architecture that drives the OCbotics approach. As hinted at in Figure 10.1, layer 3 mediates the user's goals vertically to all system layers below and horizontally to all OCbotics individuals of the system.

10.5.1 Immersive Swarm Control

The preceding examples of web-weaving quad-rotors in Section 10.3 and collaborative surface maintenance swarms in Section 10.4 implement spatial operations. To some extent, both culminate in the tandem of local cues and resultant trajectories. As a consequence, defining spatial targets for arbitrary subsets of a swarm deems to be an adequate task generalisation for a first prototype of a level 3 user interface. A " human-in-the-loop" system design forces one to

clearly define the level of influence a user may exercise versus the level of autonomy the system may keep [67]. Therefore, we elaborate on the different levels of access implemented by our prototype right after we outline its technical foundation.

Technical Setup

Focussing on interactivity, we decided to utilise the turnkey infrastructure of one of the comprehensive game and simulation engines. In particular, we decided to use Unity [475] as it provides a very shallow learning curve (compared to its competitors) while still providing a powerful coding infrastructure that allows to write custom plugins in C# and which offers a wide range of third-party plugins in a dedicated asset store. Aiming at the implementation of a high-level interface, we tapped into these resources as much as possible and bought, for instance, commercial code bases for simulating flocking behaviours [476] and automated path finding in three-dimensional environments [477]. We further built on Unity demos and plugins that support current hardware solutions such as the Oculus Rift head-mounted display [478] and the Razer Hydra motion controller [479]. In combination, these hardware solutions allow us to emulate an augmented reality scenario for controlling an OCbotics swarm. Figure 10.8 shows the model of an OCbotics swarm being setup in Unity3D. The light green circles depict waypoints computed by the path finding algorithm, the dual-view perspective at the bottom-left corner of the screen indicates the current view of the attached head-mounted display. The bottom-right window displays the library of components used for modelling the scene, the list at the right-hand side of the screen shows the components that already constitute the scene.

Behavioural Control

Our user interface prototype immerses the user into a virtual reality shared with the OCbotics swarm. In the long run, the simulated swarm is meant to make way for a real one, and the virtual reality for an augmented reality. Already, the user can observe the whole swarm or a subset tracking it with a virtual camera that follows in a distance and which aims at the centre of the set of selected individuals. The user can exercise control on any subset of the swarm, hence he may direct flocks of individuals or single individuals at a time. The interface provides all kinds of state information about the selected individuals, such as (averaged and variance of)

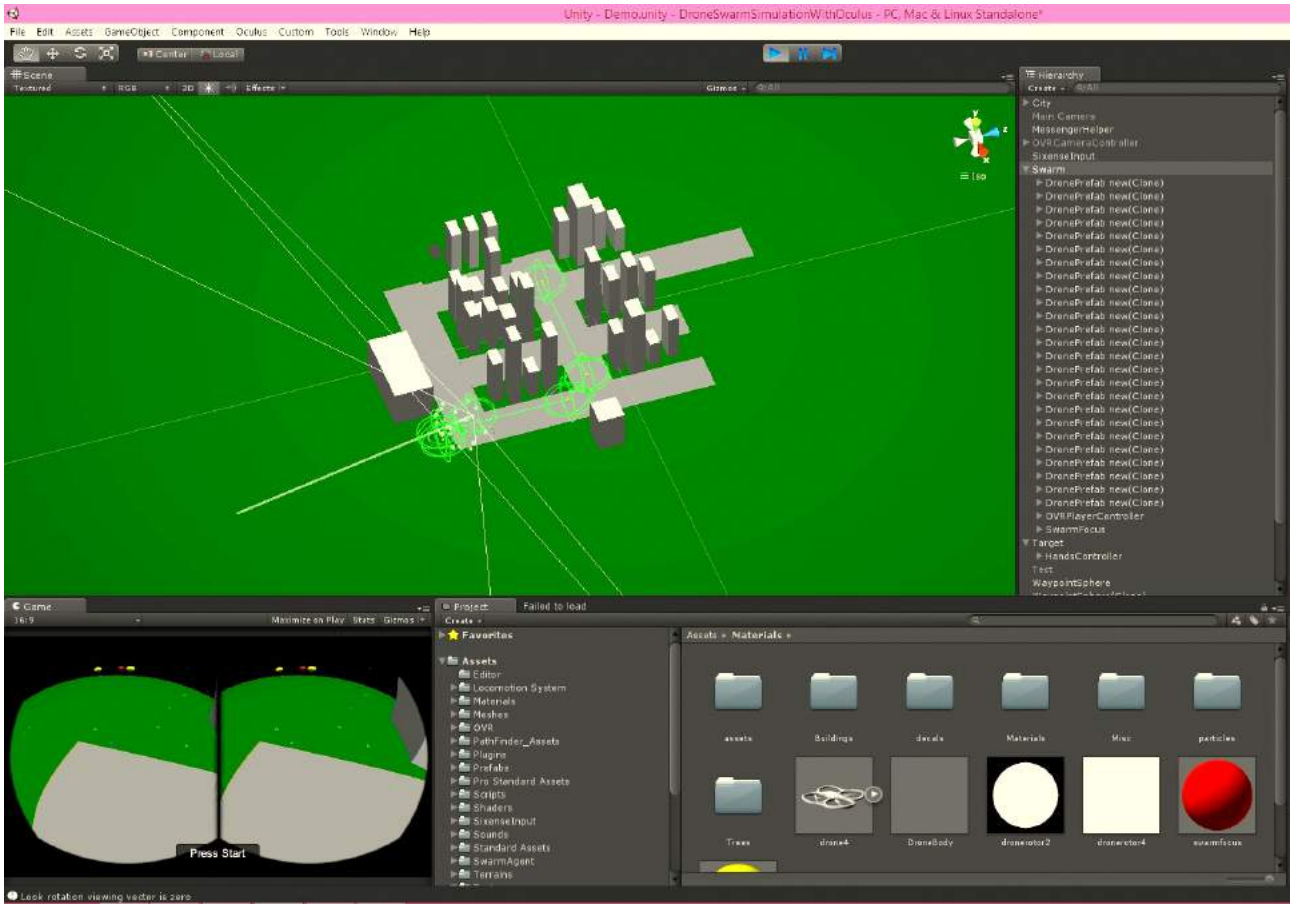


Figure 10.8: OCbotics Swarm Modelled in Unity3D. The Unity3D environment allows us to integrate complex simulation models and immersive user interaction hardware such as motion-based input controllers and head-mounted displays.

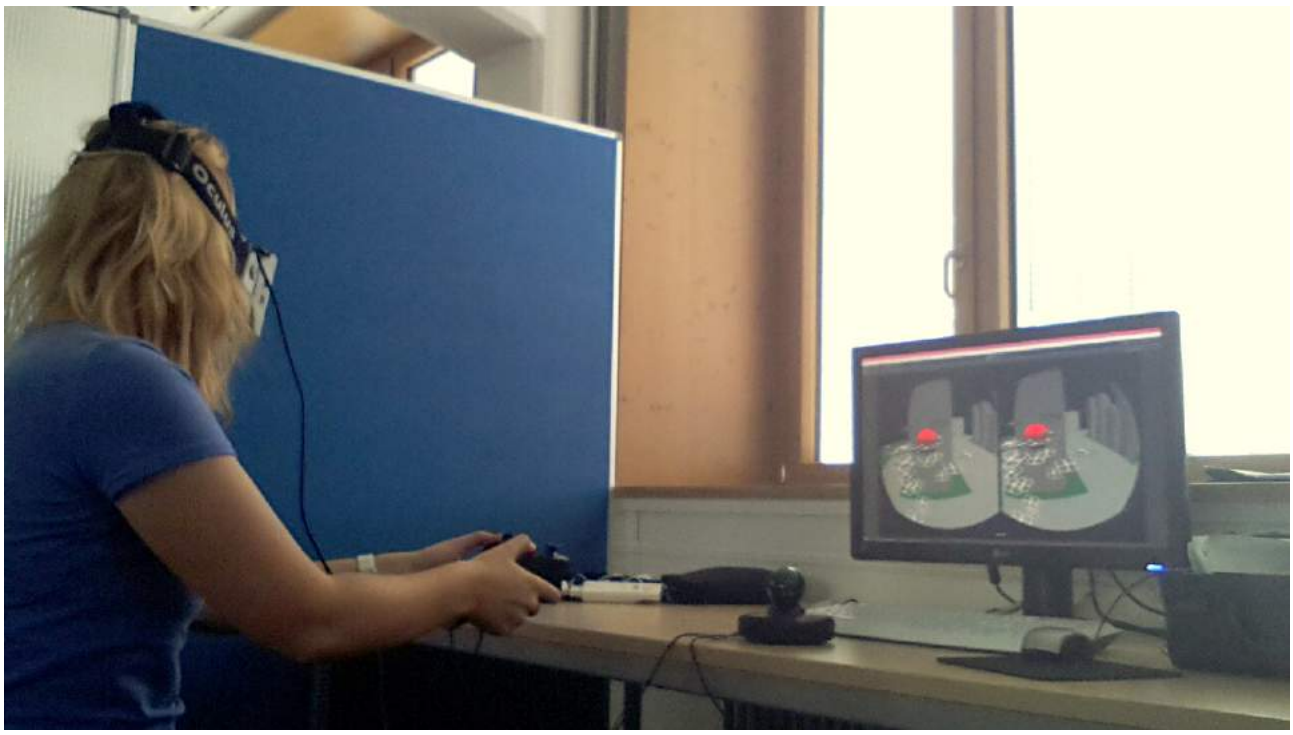


Figure 10.9: Simulated Immersive Swarm Control. One of the authors is immersed into an OCbotics simulation. She navigates through movement of her head and using a continuous joystick of the two motion-controllers. The pair of controllers empowers her to (literally) draw new spatial relations between the simulated objects, e.g. to set new targets for subsets of the swarm.

remaining battery life, current target, current trajectory, and currently perceived neighbours. The user may switch between individuals and greater subsets of the swarm by simply selecting them. Next, he may change the target of flight or even individual control points along the way. Of course, he may also change the parameters of the selected individuals such as their urge for alignment. In our prototype, the user is immersed into the scene of the simulated swarm (see Figure 10.9) so he can easily trace its activity, understand its relationship to the current target and to obstacles, and to rectify it, whenever necessary.

10.5.2 Semi-automated Control between Exploration and Exploitation

The presented simulated prototype for immersive swarm control shows how high-level goals such as setting a new target of the swarm can be communicated in an intuitive way. Differentiated selection of swarm individuals as well as setting local attributes, such as local targets or local waypoints, are simple yet clear examples of moving from abstract, high-level goal descriptions

(target/swarm) to specific low-level commands (trajectory waypoints/individual). For a swarm and and individual to reach the specified targets or waypoints, complex calculations have to be performed. In the given example, the need to avoid obstacles and to find optimal paths as well as the coordination among swarm individuals on their way are outsourced to third-party plugins [477, 476]. In the general case, also considering other tasks communicated on layer 3, the necessary behaviours could evolve in sandboxed simulations (layer 2) and be optimised based on local performance feedback (layer 1).

10.6 Conclusion

In this paper we have introduced OCbotics as a comprehensive approach to designing swarm-based, self-organising robotic systems. OCbotics is driven by a multi-layered observer/controller architecture that allows to optimise and adapt an adaptable system. Adaptation is required in order to maintain or increase the performance exhibited by the system under observation and control—either by optimising or extending existing behaviours, or by innovating, i.e. generating, simulating, and optimising novel behaviours. The performance, in turn, is measured in terms of user-defined goals which may also change over time. We have presented three different projects that operate at different levels of the discussed architecture: Web-weaving quad-rotors with an emphasis on optimised local reactive behaviour, evolution of collaborative behaviour to efficiently work on surfaces, and an immersive user-interface for setting and changing user-defined goals. While the three examples slightly vary regarding their applications, they are connected through the common themes of self-organisation, rule-based behaviour, and adaptation, and of course, the O/C architecture to host them all. With the pieces of the puzzle at hand, the next obvious step is to put them into place, to forge the software components into one (if heterogeneous) code base, to connect the layers of the architecture, to develop a repertoire of recombinable goal definitions, and to transfer the partially still virtual implementations of all levels onto an actual OCbotics infrastructure.

Part III

Modelling Interactive Self-Organising Systems

Chapter 11

CoSMoS in the Interactive Simulation Curriculum

Animations at first, then real-time computer graphics and human-computer interaction techniques have made interactive simulations possible. Nowadays, they play an enormously important role in training the operation of complex technology such as aircraft, and they have achieved a remarkable share in the computer gaming industry. The fast emergence of virtual and augmented reality solutions promises an even wider spread and a greater impact for interactive simulations in the near future. Due to the multifaceted nature of interactive simulations in terms of confluent scientific fields, due to the underlying iterative and agile development processes, and last but not least due to the inherently central human factors, we have been integrating the CoSMoS process of complex system modelling and simulation into our course curriculum on interactive simulation for computer science graduate students. In this work, based on an overview of the contents and the logistics of the course, we present our conceptual efforts towards this goal. We emphasise the role of the CoSMoS process, discuss its impact on the students' projects, and we provide concrete examples.

Sebastian von Mammen, Sarah Edenhofer, Jörg Hähner. *CoSMoS in the Interactive Simulation Curriculum.* In: *Proceedings of the 2015 Workshop on Complex System Modelling and Simulation (CoSMoS)*, 2015, pp. 85–106.

11.1 Introduction

Since the first human-in-the-loop simulators entered the market in the 1980s [67], interactivity has evolved into an increasingly important aspect of scientific simulations. Nowadays, established mathematics frameworks such as Mathematica, Maple or Matlab provide ample support for visualisation routines and interactive parametric exploration of any devised models, whereas development frameworks such as Unity3D, Unreal Engine or CryEngine that primarily target the computer gaming market are offered and marketed in the context of simulations as well. Due to the fast-paced strides towards ubiquity of virtual and augmented reality systems [480], for instance by utilising widely availably smart phones, we expect an even more accelerated spread of interactive simulations in the near future.

Numerous areas of computer science feed into the development of an interactive simulation—human-computer interaction, real-time computer graphics, visualisation, modelling and simulation approaches, etc. The according methodologies and techniques are deployed to make a simulation model accessible to the user. In addition to translating reality into an adequate domain model and further into a suitable computational representation, or platform model, the creators of an interactive simulations are confronted by an abundance of user-related interfacing challenges. In all brevity, they need to translate the user's wishes into effective commands of control and model changes, and they need to translate the matter-of-fact results of the simulation process into visualisations (mostly), that are quickly understood and capture rather than lose the user's attention. To render the trade even more challenging, all of these translations need to happen at rather high rates that provide for an uninterrupted interaction experience.

Motivated by their great and growing importance, we set out to teaching students foundational knowledge about interactive simulations. In particular, we designed a university course to empower computer science graduate students with an interwoven in-depth apprehension of methods in the associated fields. Thus, the students acquire knowledge to evaluate and skills to contribute to the design and the programmatic implementation of interactive simulations. In this work, we present our course concept, focusing on the role of the CoSMoS process of complex system modelling and simulation. Based on a description of our course concept (Section 11.2), we highlight the role of the CoSMoS process in the curriculum as a whole, and with respect to the accompanying student projects, in particular, in Section 11.3. Next, we present several

select student projects (Section 11.4), also shedding light on the development processes the student went through throughout the term. We conclude this work with a summary of our findings and an outlook on future work on CoSMoS for interactive simulations.

11.2 Synopsis of an Interactive Simulation Course

An interplay of a variety of computer science disciplines provides the foundation for interactive simulation. Accordingly, the contents of a course on interactive simulation greatly vary dependent on the expected knowledge base of the students as well as complementary courses offered by the hosting institution. In our case, we devised a university course suitable for master students in computer science and closely related programmes of study. The course runs for four months, staging a 2-hours-lecture and a 2-hours-tutorial each week. In combination with the allotted project work, the course demands for a total workload of 150 hours.

In the following paragraphs, we summarise the contents of nine provided lecture units. After an introduction to the subject matter, we teach the CoSMoS approach to modelling and simulation. Next, foundations of computer graphics are conveyed, as well as a mathematical display of real-time physics computation models and algorithms. Visualisation methods and an introduction to human-computer interaction techniques complete the first block of basic lecture units.

The second block of advanced lecture units focusses on model representation and process optimisation both of which are important constituents of interactive simulation technology. After presenting the foundations of discrete event simulation and an array of computational representations, popular conservative and approximative acceleration mechanisms in the realm of interactive applications are discussed. As the versatility and the transferability of an agent-based modelling (ABM) approach is rather unique but can easily result in costly computations, we commit another lecture unit to introducing novel research concepts that promise to scale ABM to reach interactive performances.

11.2.1 A Short History of Human-in-the-Loop Systems

The history of interactive simulation begins with efforts to enhance existing simulation data by means of interactive custom animations. We present an according example, a SIMAN job shop

simulation model of an automatic guided vehicle system visualised by the CINEMA animation system [481]. The optimisation of industrial workflows was the most compelling argument for such animation systems in the 1980s. At the time, the market offered an array of simulation animation tools, including Model Master, XCell, and Performance Analysis Workstation. Next, solutions were offered that tightly coupled interactive visualisation with the underlying simulation. See-Why was one of these packages that promised Visual Interaction Simulation (VIS). An according example allowing the configuration of a locomotive servicing centre is shown [482]. Definitions of basic terms such as model [483], simulation [27], and the early-conceived notion of interactive simulation (‘ on-line simulation’) [65] follow the introductory historic examples. The distinctive feature of interactive simulations is the possibility of human influence during the simulation process, typically referred to Human-in-the-Loop systems [67]. We look at the taxonomy of interactive simulations, their advantages over stand-alone simulations, established fields of application, technological challenges, and their historic evolution in respect to programming paradigms, languages, and interfaces. The basic steps taken in a simulation project, especially under consideration of interactivity, and several examples of state-of-the-art interactive simulation systems round off this lecture unit. The examples are organised to guide the students from comprehensive immersive solutions with special hardware configurations (driving and flight simulators) to software-only solutions, which are the focus of the lecture.

11.2.2 The CoSMoS Process & Gamification

The orthogonal relationship between descriptive and defining models precedes the remaining contents of this lecture that primarily aims at the process of modelling and simulating complex systems. Examples for seemingly disparate approaches are provided that put different weights on these respective modelling purposes: In detail, these are understanding complex behaviours of real-world systems, simulating complex system themes, engineering complex algorithms, and engineering complex systems [484]. We consider the means of scientific instrumentation (extrapolation, conversion, augmentation) to become aware of its limits and limitations [485] and to define the products of the CoSMoS modelling and simulation cycle. The definition of these products motivates an elaborate discussion of the phases of discovery, development and exploration [15]. Interactive simulations need to engage their users. The relatively novel paradigm of turning burdensome chores into games suits this challenge well. Hence, we proceed with

the presentation of game definitions and, more specifically, aspects of development of computer games [486]. The short history of serious games (starting in the early 2000s) is summarised [417, 487] and representative examples are demonstrated (e.g. [488]). Their concrete successes in terms of engagement are discussed and a comprehensive list of game design elements [416] is presented that can be utilised to 'gamify' an interactive simulation. All of these elements can be derived from the cornerstones of intrinsic motivation, namely relatedness, competence, and autonomy which are explained as well [427]. In the context of interactive simulations, these aspects can be considered during the discovery phase, whereas gamification typically takes place during the development phase of the CoSMoS process.

11.2.3 Computer Graphics Foundations

The increasing availability of dedicated graphic processing units (GPUs) promotes the utilisation of a standardised 3D rendering pipeline for any kind of visualisation needs, whether 2D or 3D, vector-based, or otherwise. Therefore, this lecture units seeks to empower the students with a basic understanding of this rendering pipeline [489, 486, 490]. Basic concepts that are presented in this lecture unit are: object definitions based on geometric primitives and various kinds of textures, the view reference, spatial transformations (also introducing quaternions in the context of rotational operations), basic types of lighting, shading, and light sources, shadow definitions and different implementation techniques such as shadow maps. A short walkthrough of generating 3D graphic assets suited for real-time rendering rounds off this lecture unit.

11.2.4 Real-time Physics

This lecture seeks to provide a solid grip on real-time capable approaches to simulate physical processes [491, 492]. We have a quick look at the taxonomy of the vast field of physics simulation [319] but we focus on real-time methods of forward dynamics, covering three categories: rigid-body dynamics [320, 321], soft-body dynamics [322], and particle physics [323]. Next to the general laws of motion, we look at non-penetration constraints, collision resolution and friction forces, and complementary constraints in the context of rigid body simulation. For calculating the respective forces, we present the penalty force method, Lagrange multipliers, impulse-based simulation, and reduced coordinate formulation as well as the Coulomb friction model.

We introduce a taxonomy of constraints and explain how they can serve as representation of mechanical joints of articulated bodies [324]. We follow the steps to transform the resulting differential algebraic inequalities into an efficiently solvable linear complementary problem. We conclude the integration of forces with a brief recap of basic methods of numeric integration, starting with Euler and Runge-Kutta. We discuss algorithms for efficient collision detection and contact point generation, e.g. [493, 327]. We then widen the scope of this lecture unit, looking at one specific approach to computing incompressible deformable mesh dynamics that is superior to alternative approaches in terms of efficiency and accuracy [322]. Finally, we introduce to real-time particle physics, explaining particle approximation functions based on the notion of kernel functions [323], culminating in recent advancements in unified real-time physics simulation [172].

11.2.5 Visualisation Methods

To a large extent, interactive simulations imply some kind of visualisation of the underlying models and the emerging simulation processes. In this lecture unit, we emphasise the necessity to consider human perception and information processing when crafting the platform model of an interactive simulation and we provide an overview of foundational visualisation techniques. We follow the structure of numerous textbooks on this subject matter and motivate the discussion on the human vision apparatus by providing several examples of optical illusions [222]. In a 7-step guideline, we establish an idea of the selection of the proper visualisation method embedded in the context of data acquisition and the intended modes of interactions [494]. Qualitatively, visuals can be measured in terms of novelty, informativeness, efficiency and aesthetics [103]. We shed light on various scientific, multidimensional, multivariate visualisation methods [495], before we turn to visualisation techniques that allow for the immersive augmentation of simulation contents, such as examples of flow visualisation [496], graph-based visualisations [497], or the transformation of volumetric (4D) data into 3D surfaces [498].

11.2.6 Human-Computer Interaction Techniques

The design of interactive simulations necessitates an interface between human and computer. This lecture unit provides the necessary background, starting with a brief history of HCI re-

search [499]. Human interaction requires the user to process and translate sensed information into motor activity [500]. In general, user interactions can be classified as operations of selection, manipulation, navigation, and system control [486]. As any other design task, the design of interactions is the result of a tradeoff between multiple goals and constraints [20]. We present a top-down approach to designing interaction scenarios that starts with the definition of an application's requirements and arrives at individual interaction tasks. We have a brief look at multimodal approaches, e.g. affective, perpetual, attentive, and enactive interfaces [501], and we explore the modes of interaction of an embedded multimodal prototype game [502]. We quickly step through established and emerging immersive hardware technologies, including devices of motion sensing and object tracking capabilities. We convey a general understanding for the hard latency limitations of interaction hardware and we provide recipes for rather general issues that arise in real-world sampling, i.e. noisy sensing and the state estimation problem [490, 486].

11.2.7 Discrete-Event Simulation

In this lecture unit we provide an overview of computational representations as well as modelling and simulation approaches. We start out with explaining the basic terminology of discrete-event stimulation (DES) in the context of previous lecture units, especially those described in Sections 11.2.1 and 11.2.4 [503]. To this end, hybrid simulation and combined simulation concepts are of great importance. We roughly trace the history of this seminal field in terms of DES software packages and languages [504]. Three 'world views' on DES (event scheduling, activity scanning, and process interaction) serve as the starting point for our venture into historic approaches. We meticulously describe the elements of a DES and provide a glimpse at charts already used for engineering DES back in the 1960s. These diagrams (activity cycles, wheel chart, flow charts) are our point of departure towards other computational representations commonly used for modelling and simulation: Finite state machines, UML transition diagrams, Petri nets [505], artificial chemistries [337], cellular potts [506], cellular automata [507], random boolean networks [341], boids [399], L-Systems [508], swarm grammars [3], and the general approach of agent-based modelling [55, 53].

11.2.8 Acceleration Algorithms

Low latency requirements of the interaction interface (Section 11.2.6) as well as the desire to serve large, complex models for interactive exploration at real-time demand for the utilisation of sophisticated acceleration algorithms. Computer graphics and real-time physics are currently occupying this niche and this lecture unit aims at exhibiting their commonly used, highly efficient approaches [489].

It is divided into four parts: First, we focus on bounded volume hierarchies (BVHs) and binary space partitioning trees (BSPs). While BVHs are built bottom-up based on bounding volumes that enclose geometries or other bounding volumes recursively, BSPs are generated top-down by recursive division of the simulation space. We also provide guidelines to coping with (a) mobile and (b) deformable objects [509, 510, 511]. Second, we explain different culling techniques which ensure that graphical objects are not pushed through the rendering pipeline (Section 11.2.3), if their contributions to the final rendering are marginal or not existing. Examples are surfaces that lie outside of the view frustum, those that are hidden behind objects, or those that are so far away from the camera that they could hardly be seen on the screen. Third, we present approaches that lower the level of detail (LOD) of the rendered objects to the match the actual needs—as opposed to always rendering at the highest possible level of detail [248]. An example of LOD is the number of triangles of discrete geometries which can, for instance, be selected according to the distance to the camera. We conclude this lecture unit showing stochastic acceleration algorithms for collision detection.

11.2.9 Dynamic Model Abstraction

Motivated by the outlook on large, multi-scale, multi-representation simulations [287], we tackle the issue of ever-growing computational complexity by means of dynamic model abstraction techniques in this lecture unit. Particularly, we focus on adaptive optimisation of agent-based models, as they can serve as a generic computational representation. Concerning the immense computational costs running large-scale simulations, we discuss the limitation of different model aspects and how they could improve efficiency. We conclude this investigation with the realisation that if we want to model and compute natural systems, we need to consider dynamic

systems with dynamic interaction topologies (D^2S) [19]. In addition to hardware-based solutions (e.g. [512]), we promote dynamic model adaptation. Agent compression identifies and subsumes clusters of similar agents [513]. The dynamic extension of this approach considers container agents to maintain similar agents and to offer the possibility to remove or add individuals on demand. Compression managers are responsible for (a) organising the container agents and their contents, and (b) representing the compressed agents to the remainder of the model [514]. Taking this idea even one step further, we provide the detailed steps of the self-organised middle-out abstraction approach [268] and we show its capabilities with respect to a decentralised, agent-based blood-coagulation simulation [515, 516].

11.3 CoSMoS' Central Role

The CoSMoS process is introduced right after a general introduction to the course (see Section 11.2.2), as it provides a flexible, yet focussed guideline for all phases of the development of interactive simulations. In this section, we first detail a way of applying the CoSMoS process to student projects, following the explanations in [15]. Second, we present a course infrastructure to realise this approach.

11.3.1 CoSMoS for Interactive Simulation

We discuss the three phases of the CoSMoS cycle (discovery, development, exploration) in the context of interactive simulation based on the five activities performed during each phase: scoping, modelling, experimenting, documenting, and interacting.

During the *discovery phase*, the greatest challenge to the students is the primary need to settle on an application domain and to define the goals of the interactive simulation, e.g. teaching contents or providing for a scientific exploration tool. Although the students appreciate the opportunity to freely chose an application domain for their projects, they seem to be more comfortable when provided with a theme, for instance biology. The only constraints regarding their choice is the projects' evaluation based on the following aspects, which are set to ensure their usefulness and the comparability.

Science The model that drives the resultant prototype has to be scientific, i.e. it has to be based on scientifically published results. A CoSMoS compliant development process certainly supports this endeavour. In addition, the modelling domain, the validity of the modelled system, its degree of innovation, and the computational representation and algorithms used give strong indications for a scientific approach.

Gamification The prototype has to motivate the user to interact and explore the simulation space. One can try capturing this aspect quantitatively by describing interaction possibilities, user guidance, usage of game elements, and the factors of intrinsic motivation as referenced in Section 11.2.2.

Complexity Interacting with the prototype should be rewarding in itself, i.e. it should convey insights with respect to the underlying scientific model. The model complexity defines the scope of potentially educational contents, given the conveyed complexity considers the full extent of the underlying model.

Aesthetics An interactive simulation has to be aesthetic, not only to efficiently convey information to the user but also to motivate their involvement. Aesthetics can be promoted following established design principles, by utilising beautiful visual assets, and by combining them in novel ways.

Any steps towards desirable domain attributes, concrete domains, and even concrete goals and an application concept, necessitate answering questions about the projects' criticality, their limitations, and their measurable success. In the context of interactive simulations, the answers typically stress the relationship between the software and the user. The utility for the user, for instance, not only considers a final simulation result but also the benefit of proactive participation in the simulation process. Accordingly, limitations are not only considered regarding the accuracy and efficiency of the simulation but especially with respect to the degrees of freedom exploitable by the user and the quality of the communication between the user and the simulation, including aspects such as clarity and attractiveness. The modelling activity during the discovery phase is rather limited in the scope of a term-long project. Despite the abundance of scientific data accessible through online libraries and the large repositories of computational libraries and tools for numerous scientific domains, comprehending the elements and their relationships of a previously unstudied field is a rather difficult task. For this reason,

and also to provide the necessary degree of autonomy to intrinsically motivate the students, we allow the students to decide on a concrete domain and goal by themselves; based on supervisory feedback on a written proposal and classroom presentations with subsequent discussions, the core ideas can then be quickly translated into first proof-of-concept prototypes. The discovery phase is decisively shaped by documentation activity—from coarse to fine grained searches for references and tools, through merging sources, assumptions and ideas into a concept proposal that includes an early domain model, to creating a first prototype that provides evidence for the created line of argument.

During the *development phase*, documentation about the students' activities is similarly important. However, to a great extent, it coincides with the development of the platform model, an accompanying commented code base, and its transcription for a given simulation platform. To help reduce the burden that a comprehensive interactive simulation project incurs, we diminished the scoping activity of the development phase and taught about various tools of the trade for interactive simulation development—ranging from 3D asset creation over scripting and high-level, component based model compositions to utilising third-party plugins and libraries for the targeted development environments. In frequent presentation and feedback sessions, we ensured that domain elements were properly represented and domain behaviours were not directly encoded in the models. Adding instrumentation to the platform model plays an important role for interactive simulations. This step should closely follow the interaction concept developed as an extension of the usual domain model, i.e. one that encompasses the user as a special model element. Nevertheless, the targeted simulation platform may provide a rather special interaction infrastructure. For example, the ubiquity of mobile, multi-touch platforms equipped with relatively weak processing capabilities competes with the processing power, storage capacity, and extensibility of desktop systems. Clearly, any specific interaction platform demands for individual adjustments of the platform model to realise both the interaction and the simulation concept. Experimentation in the development phase begins with the first prototype supporting preliminary user interactions. At later stages of the development phase, it increasingly involves feedback from testers not directly involved in the development work. Beyond honing the visualisation, consistent design, usability and the scalability of their platform models in terms of parameter settings, numbers of interacting agents, increasing levels of difficulty and the fine line between balanced, rewarding interaction and user boredom and frustration.

During the final stage of the course project, the *experimentation phase*, the students focus on logging and analysing user responses to their simulations. To keep the amount of work at a level reasonable in the context of our course, the students are asked to try each others' simulations and to ask their friends and relatives to provide them with some preliminary feedback. This exposure typically already provides comprehensive insights into the users' general interest in the topic, their opinion about model complexity and aesthetics, and whether they think it is educational. Based on these evaluations, the students are encouraged to hone their software and to launch more comprehensive online surveys. However, these more rigorous steps are not mandatory course stipulations. Nevertheless, the gathered preliminary data in combination with the initial motivation of their projects, the development processes and the implementation results, serves as an extensive basis for fleshing out their final report. It culminates in conceptual improvement that could instigate the next development cycle.

11.3.2 Course Project Infrastructure

Above, we already touched upon the students' deliverables and how their realisation is backed by the CoSMoS process. Now, we briefly present the logistical infrastructure of the course setup to support the traversal of the CoSMoS process throughout the term.

During the first lecture, the students are first informed about the course contents and its stipulations. For the remainder of the lecture, we present and explain several examples of possible project concepts. Although the students may conceive a project idea completely on their own, providing examples proved important to communicate the expected scope and the imparted opportunities. Within ten days' time, teams of two students need to author a proposal of their projects. On two pages (ACM double-column format), the students need to motivate, present and detail their concepts. Hereby, the envisioned user experience plays an important role as it ties different aspects of the envisioned simulation together and it implicitly underlines its goal. From a CoSMoS perspective, the project proposal is part of the documentation activity of the discovery phase. As such it serves not only as a platform for the students to substantiate their initial ideas and consistently brush up their findings but also to communicate their concept to the instructors.

At the time of the proposal submission, a second lecture unit has introduced the general topic

of the course (Section 11.2.1) and a first tutorial session has familiarised the students with the development environment that we recommend (in previous years, we recommended Unity3D). The day after the submission of the proposals, the students are asked to present their concepts in short 3-minute presentations during the tutorial session. In this way, all the students in the course would gain an overview of their peers' projects and learn about new ideas, possibly even about the usage of previously unknown code snippets, etc. The quick start into the projects and presentations early in the term help the students build up momentum for their projects. In fact, until the last few weeks of the term, the students would present the state of their projects bi-weekly. This fosters a certain sense of togetherness and it ensures guidance to maintain high productivity and to avoid frustration.

Two weeks before the end of the term, final reports are due (six pages, ACM double-column format) that should ideally condense the documentation recorded throughout the whole term. One week later, the students need to submit their projects, including batches of slides for the final presentations which are given in front of faculty and students of the whole department. The audience is asked to vote for the best entry in terms of the generic project criteria: science, complexity, gamification, and aesthetics (Section 11.3.1). A 15-minute brief oral exam at the end of the term makes sure that the students have learned and understood the diverse contents of the course and their relationships.

11.4 Select Student Projects

In this section, we present select student projects that were developed in two iterations of our interactive simulation course. First, we describe some of the outcomes exemplarily. Second, we shed light on the CoSMoS-driven development process of a specific project.

11.4.1 Examples

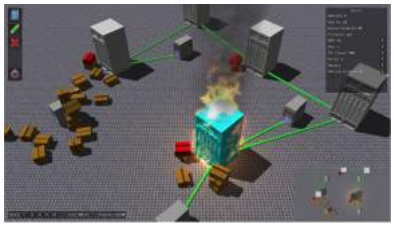
During the first iteration of the course, the majority of the students chose "technical systems" topics such as routing in communication networks, smart cars, and power networks. Figure 11.1(a)-(c) shows according screenshots. The user is tasked to build and maintain power or communication infrastructures to ensure their proper functionality. In the network routing and

the smart car example, the user also had to guide the network activity itself by laying out flow paths of the respective traffic. Some students also journeyed towards biological themes such as cellular automata as seen in Figure 11.1(d). Here, a game of life variant served as the basis for a two-person game with the goal of conquering as much space as possible solely by adjusting the cells' rules. During the second iteration of the course, we proactively advertised biological and natural phenomena as an exciting and multifaceted field to motivate the student projects—yet, they were still free to take their projects into other directions. As a result, three groups let their projects revolve around bees (we had not motivated this trend), see Figure 11.1(e)-(g). In the first one, the user had to guide a bee's waggle dance to point its peers to the location of a food source outside the hive. Figure 11.1(f) shows a screenshot of a bee simulation that focusses on the challenge of gathering nectar and thereby helping flowers pollinate. Lastly, a complex real-time strategy simulation is presented in which bees need to gather resources, maintain their hive and defend it against wasp intruders. Other examples included the user-guided migration of a flock of geese (Figure 11.1(h)) or the establishment of a fine balance of interdependent inhabitants in a simulated aquarium (Figure 11.1(i)). The interdependency of species provided the basis of yet another title where a new ant species threatens to overrun a native species and the user is tasked to maintain a balance by building barriers or proactively diminishing one or the other ant population (Figure 11.1(j)). Focussing on solitary species, a squirrel simulator offered the experience of sharing a rodent's worries: collecting, burying, and finding enough nuts to survive the winter season (Figure 11.1(k)). The importance of climate also inspired "Cloud Computing", where a user was tasked to set the environmental conditions in such a way that certain weather phenomena such as rain or tornados would emerge (Figure 11.1(l)).

The set of presented examples emphasises the flexibility of the course project in terms of contents, perspectives and goals of the student project while addressing the project requirements as outlined above (Section 11.3.1). Next, we dive into one specific project and shed light on how the CoSMoS process informed its development.

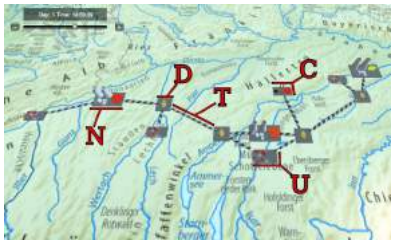
11.4.2 A CoSMic Case Study: "Drink & Drive"

One student team decided on creating a serious game about the negative effects of alcohol on traffic participants. They understood that although some accurate simulators exist for this



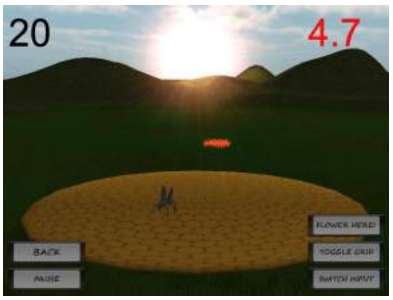
(a) Network Routing

(b) Smart Cars



(c) Power Networks

(d) Cellular Automata



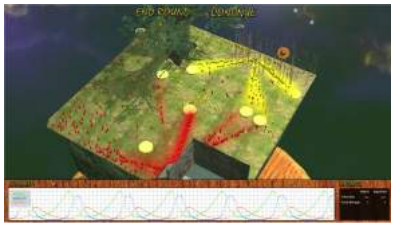
(e) Waggle Dance

(f) Pollination

(g) Beehive Defence

(h) Migrating Birds

(i) The Aquarist



(j) Invasive Species

(k) Squirrel Simulator

(l) “Cloud Computing”

Figure 11.1: Screenshots of interactive simulations developed as student projects in two iterations of the course.

purpose (e.g. [517]), they don't provide for a stimulating, engaging experience. At first the students were hesitant whether their idea was acceptable as it attempted to approach a serious topic in an engaging, fun way. We encouraged them to try anyway. Findings about games that had been developed for this purpose, such as [518], further boosted the students' ambitions. These preceding titles had disconnected from the actual problem too much, for instance by assuming a third-person perspective on the driving situation. Quickly, the students realised that their interactive simulation should fill this gap and make their title "Drink & Drive" both fun and educational, so that the target group of soon-to-be drivers and young drivers would engage in and learn about this fundamentally serious topic. The second part of the discovery phase of their project shed light on actual models of impairment of drunk drivers. Its last part posed the greatest challenge: Merging the seemingly conflicting concepts of learning about the severe consequences of drunk driving on the one hand, and the need for user engagement on the other hand. They achieved this by two means. First, they decided to represent the game itself at a level of abstraction different from the effects of alcohol. In particular, the game implemented widely-known "Mario Kart"-style game mechanics and a simple, cartoonish look (Figure 11.2(a)), whereas the impairment of alcohol was reflected by realistic effects, including the deterioration of clear-sightedness, darkening the edges of the vision, attenuating sounds, and prolonged reaction times (realised by increased simulation speed), see Figure 11.3. Second, they introduced gamification elements including timed laps and collecting high scores by picking up precious diamonds from the track (Figure 11.2(b)). However, fundamental game mechanic to engage the users was invented later during the experimentation activity of the development phase. The students laid out the development phase very professionally and, together with the other students, received bi-weekly feedback to stay on track. Knowing that experiments could yield the key to an engaging user experience, the students tested various parameter settings of the driving model, its reactivity to the user input, as well as different interaction modes between the steered vehicle and the environment. From what they learned they were able to invent a mechanism to ensure a challenging and well-directed user experience. In particular, they translated the idea of collectibles on the track to their application domain and positioned beer cans at certain locations (Figure 11.2(b)). Their uptake would increase the blood alcohol level and driving would be impaired. The impairments would render it difficult to complete a track within a certain amount of time. Given the mechanics of driving, impaired driving, high-scores and time-laps, the students just needed to find the right balance to finish the

development phase of their simulation. "Drink & Drive" was voted best entry in the public presentations at the end of last term's interactive simulation course. In addition, it stirred a lot of excitement when it was offered for play as part of the Girls' and Boys' Day at our university. Based on these successes, the students feel that the most fundamental aspect that could drive a second development cycle would be the port of "Drink & Drive" to mobile devices for reaching a greater audience.

(a) (b)

Figure 11.2: (a) A first-person default view is reduced to a simple steering wheel dashboard and a few icons that represent the time left to complete the track (the heart icon in the upper-left corner), the achieved score (the diamond icon next to the heart icon), and the alcohol blood concentration (to the right-hand side). (b) Alcoholic beverages and diamonds can be picked up from the road - the first increases the driver's blood alcohol concentration, the latter his score.

0.0‰ 2.0‰ 3.3‰

Figure 11.3: The alcohol blood level directly translates to impairments of vision, hearing, and reactivity.

11.5 Conclusion and Future Work

In this paper, we presented an experience to adapt, teach and apply the CoSMoS process in a graduate computer science course on interactive simulation. We first laid out the multifaceted synopsis of the course before elaborating on the central role of the CoSMoS process in the context of the students' term-long projects. Finally, we briefly presented some of the results of the students' works and expanded on one of them, exemplarily. The scientific claim, the notion of self-organising processes with a focus on the interaction of numerous interwoven parts, as well as the agility of the CoSMoS process lend themselves well for backing interactive simulation projects.

Although both the results and the students' feedback have been rather encouraging regarding the course contents, its layout and its general methodology, we are eager to further improve several aspects. It might, for instance, be beneficial to have certain activities of the different phases of the CoSMoS process take place in groups during the tutorial sessions. Scoping during the discovery phase has repeatedly proven difficult to students. An experienced teacher could guide the process and ensure that multiple options are considered by each group. More generally, we believe the CoSMoS process could still be more tightly integrated in both the lectures and the tutorials, by providing an outlook of its application to the lecture units' contents. For instance, one could illustrate the application of the CoSMoS phases not only to the project as a whole but also to individual aspects such as computer graphics and visualisation—from the goals and ideas of the used assets, the designed environment, over their creation and programming to experimenting with their parameters.

So far, we have not considered building on the CoSMoS process for evaluating the students' works or their performances during the exams, except for considering CoSMoS-supported project criteria (Section 11.3.1). Yet, the students frequently utilised the structure of the process for classifying and presenting their work. In particular, they frequently referred to its phases and activities during their bi-weekly oral presentations and let their final project reports revolve around them. Hence, one research question that remains is whether and to which extent the individual phases of the CoSMoS process could be coupled a priori with the students' evaluation.

Last but not least, the CoSMoS process could be expanded to even better accommodate the development of interactive simulations. As they are typically designed for learning and training, an according 'engagement model' could, for instance, be an additional, desirable product of the discovery phase, complementing the domain model. It could comprise learning targets, explicitly visualised versus implicitly utilised data, the tasks and mechanics of the interfaces provided for interacting and exploring the domain model, as well as means of motivation, such as gamification elements. In combination with the domain model, such an engagement model would provide for a clear conceptual foundation for the development phase.

Chapter 12

A Graph-based Developmental Swarm Representation & Algorithm

Modelling natural processes requires the implementation of an expressive representation of the involved entities and their interactions. We present *swarm graph grammars* (SGGs) as a bio-inspired modelling framework that integrates aspects of formal grammars, graph-based representation and multi-agent simulation. In SGGs, the substitution of subgraphs that represent locally defined agent interactions drive the computational process of the simulation. The generative character of formal grammars is translated into an agent's *metabolic* interactions, i.e. creating or removing agents from the system. Utilizing graphs to describe interactions and relationships between pairs or sets of agents offers an easily accessible way of modelling biological phenomena. Property graphs emerge through the application of local interaction rules; we use these graphs to capture various aspects of the interaction dynamics at any given step of a simulation.

Sebastian von Mammen, David Phillips, Timothy Davison, and Christian Jacob. *A graph-based developmental swarm representation and algorithm.* In: *Proceedings of ANTS 2010, 7th International Conference on Swarm Intelligence, Series: Lecture Notes in Computer Science*, vol. 6234, Springer Verlag, 2010, pp. 1–12.

12.1 Introduction

We are interested in modelling complex biological systems at various levels of scale, i.e. from the biomolecular level [519] to cells [304] to systems [384], etc. Different levels of resolution often require different computational techniques, such as differential equation solvers to compute physics or fluid dynamics, or engines that execute high-level agent behaviours that implement rich interaction policies and complex strategies [472]. Independent of the specific computational approaches that drive the simulation processes, they all rely on state changes, the principle of digital computation. Furthermore, a system's state determines the introduced changes, probabilistically or deterministically. This idea is emphasized in numerous computational representations such as Markov chains [520] or cellular automata [299]. The state of a system is generally understood as the states of all its subsystems including their interrelations. Consequently, states and relations are interchangeable terms that provide the condition for change, or the antecedent for a consequent in a simple *If-then rule*. A set of probabilistic rules (like in Markov chain systems) works well to represent the activities of decentralized, self-organizing swarm agents [229, 288, 230, 384, 409], including swarm-based developmental systems [378, 3].

Rule-based swarm systems seem to be a good fit to capture biological models. However, there are several hurdles that make it hard to deploy swarm models in fields outside of computer science. (1) The predicates and actions that drive the simulations—e.g. the detection of a chemical signal or the deposition of a particle—depend on the modelling domains and are usually re-implemented for different experiments. Still, many of these operations can be abstracted, parametrically adjusted and reused in different contexts. The integration of these operations into a rule-based formalism also makes it possible to utilize functionality from various computational engines such as physics engines or general differential equation solvers within one modelling framework. (2) Depending on the degree of specificity of a rule's condition and its associated actions, a theoretically simple interaction can result in an over-complicated representation. A graphical description of the predicates and the associated actions can amend this issue. (3) As swarm simulations often exhibit complex behaviours, little details—for example the order of execution and the discretization steps in a simulation—can greatly influence the outcome. Therefore, we think it is crucial to design models based on a unified algorithmic

scheme.

We have devised *swarm graph grammars* (SGGs) to alleviate some of the challenges discussed above. SGGs provide a graphical, rule-based description language to specify swarm agents and a generalized algorithmic framework for the simulation of complex systems. Fundamental operations such as creation or deletion of programmatic objects, as provided by formal grammars, are part of the SGG syntax. Through SGGs we can capture (metabolic) functions at multiple biological scales, i.e. the processes of secretion and diffusion [301], or consumption/removal and production/construction [385], respectively. As a consequence of the graph-based syntax, SGGs capture the simulation in a global graph at each computational step. Thereby, the continuous re-shaping of an interaction topology of a dynamic system is traced and interdependencies that emerge over the course of a simulation are graphically represented.

The remainder of this paper is organized as follows. In Section 12.2, immediately relevant work in the respective areas of research is presented. Section 12.3 details swarm graph grammars (SGGs) and their constituents, i.e. swarm individuals, graph grammatical rules, and a general SGG algorithm. Section 12.4 shows how the SGG formalism is applied in a step by step manner to retrace a simple boid simulation, wasp nest construction, and directed cell growth and proliferation. We conclude with a summary and an outlook on possible future work.

12.2 Related Work

Cellular automata (CAs) can be considered the first computational developmental models [340]. CAs revolve around state-based interactions of individuals given a fixed interaction topology. However, in the emerging discipline of computational developmental systems, the focus shifted towards constructive expressiveness and thus overshadowed the idea of individual-based modelling. In this section, we briefly review the emergence of CDMs and demonstrate their reunion with agent-based modelling.

12.2.1 Complex CDMs

Giavitto et al. summarize several approaches to computational developmental models [18]. The most simple ones are considered to be *dynamical systems* with sets of state variables determining

their global states. *Structured dynamical systems* are more complex; they are dynamic systems that can be divided into subsystems. Finally, there are *dynamical systems with dynamical structures*, abbreviated as $(DS)^2$-systems, for instance a "developing multi-cellular organism" [354]. In addition, Giavitto et al. describe developmental models as tuples of topology and formalism. L-systems [392], for instance, describe how individual elements of sequences are substituted in parallel. *Group-based data fields* (GBF) [113], on the other hand, operate on sets of units that are connected with a homogeneous, fixed topology not unlike cellular automata [340]. *Map L-systems* [521], similar to *random boolean networks* (RBNs) [522], promote combinatorial topologies on the interacting, or growing, data structures. There are also formalisms that explicitly integrate the topology of the modelled systems, such as *membrane computing* (MC), or *P systems* [523]. P systems draw their inspiration from membrane structures of cells, neural cells and tissues. In a more generalized fashion, *graph grammars* [524] are a means to integrate topological information into any kind of developmental model. Examples are *multi-scale tree graphs* (MTGs) and the *modèle géneral de simulation* (MGS) that represent changes of *topological collections* of units by *transformation paths* on a symbolic notation [17].

12.2.2 Graph-based CDMs

Kniemeyer et al. have developed *relational growth grammars* (RGGs) which promise, like MGS, to be a universally applicable representation of CDMs [525]. They use RGGs as extensions of parametric L-systems with object-oriented, rule-based, procedural features. In fact, modelling CDMs by graph grammars, like in RGGs, allows for the expression of all developmental data structures commonly used in the computational sciences: multisets, strings, axial trees, and relational structures (edge-labeled directed graphs). Graph grammar-based CDMs can therefore be considered as a universal modelling language, able to simulate standard L-systems, artificial chemistries and ecological systems alike. Kniemeyer et al. successfully applied the RGG model to grow multi-scale models of plants integrating their structure and function [175], and, recently, to grow architectural models [176]. They also suggested that RGGs could support *agent-based* modelling—by interpreting nodes as agents, edges as inter-agent relations, and by driving their interactions through sub-graph substitutions [174].

Almost 20 years before Kniemeyer presented RGGs, Culik et al. had extended L-systems with the means to describe plants through graph structures and their growth through graph

grammatical substitutions, which were later on referred to as *graph L-systems* [526]. Shortly afterwards, Nagl investigated the relationship between graph grammars and graph L-systems, concluding that graph grammars can be reduced to graph L-systems and vice versa [527]: identical graphs can be achieved by either sequential graph grammar productions or by parallel subgraph substitutions as realized in graph L-systems. About another decade later, Lindenmayer argued that relying on maps instead of graphs bears many advantages, e.g. a clear method for mapping between the abstract representation and the natural, growing structures and better performance due to the avoidance of transformations of the representations [528].

Recently, Tomita et al. have presented *graph rewriting automata* [529], in which lattice-based CAs evolve into complex networks through the application of production rules that change local connectivities. Sayama et al. went one step further and considered the local states of a CA to inform the development of *generative network automata* (GNA) [530].

12.2.3 Swarm-based CDMs

Developmental systems can be simulated by means of agent-based, decentralized models that incorporate diffusion of molecular signals paired with particular protein or cell behaviours [342]. A generic formalism for agent-based models was provided by Denzinger et al. [54, 55] in which an agent is represented as a quadruple $Ag = (Sit, Act, Dat, f_{Ag})$. An agent Ag can find itself in any of the situations expressed in Sit. It can perform the actions described by the set Act. Its internal data areas, i.e. local variables or memory cells, are determined by the set of possible values Dat. Based on the perceived situation and its internal data values, the agent determines the next action through a decision function $f_{Ag} : Sit \times Dat \rightarrow Act$. This representation is very expressive and follows the descriptive methodology of many natural sciences in which the principle of local cause and effect leads to associated emergent phenomena of interest.

Based on these ideas, we have introduced *swarm grammars* (SGs) that merged L-systems with an agent-based modelling approach [3]. In swarm grammars, decentralized swarm agents, or individuals, have the ability to perceive and act in accordance with Denzinger et al.'s agent definition. In particular, SG individuals can react to their local environment, differentiate, reproduce, and create structures by depositing construction elements. Albeit the fact that SGs merge several instrumental biological concepts of developmental, non-linear interaction

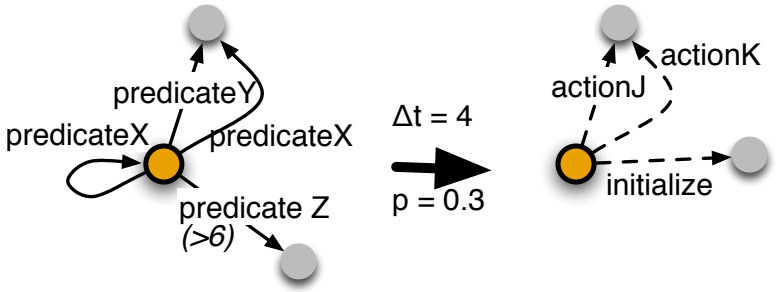


Figure 12.1: An SGG rule that queries the reference node itself, other individuals and sets of interaction candidates, to interact with them, delete some and to initialize a new node.

systems, they do not provide a unified, easy-to-use representation and algorithm that allows for systematic deployment in other scientific disciplines (as discussed in Section 12.1).

12.3 Swarm Graph Grammars

We present *swarm graph grammars* as a unified modelling and simulation framework for swarm-based systems that addresses the challenges outlined in Section 12.1, and provides a unified, graphical, rule-based modelling language for swarm individuals and a generalized simulation algorithm. The graphical description renders model dynamics more tangible and translates local interactions into global, continuously changing interaction networks. We believe that investigations into the development of these networks, in turn, could reveal quantifiable measures about *emergent* global phenomena. We address Lindenmayer's concerns about the inefficiency of graph-based CDMs by a minimalist subgraph matching procedure that only considers star networks of depth 1 around the corresponding, active reference agent.

12.3.1 Representation

An SGG agent's behaviour is described by a set of rules (Figure 12.1). Each rule tests a set of predicates (solid edges on the left-hand side) and executes a set of actions (dashed edges on the right-hand side) in respect to the acting agent itself (reference node) or other agents. Nodes represent individual agents or sets of agents. In Figure 12.1, the acting agent is displayed as an orange node with a black border. Other agents or agent groups are depicted as grey nodes. The application of the rule is associated with a frequency and a probability. Sets of predicates can attempt to identify an arbitrary number of agents. The relative location, i.e.

the two-dimensional coordinates, of the node on the left-hand side of the rule is matched with its appearance on the right-hand side of the rule. If a node does not reappear on the right-hand side, it implies that its corresponding agent has been removed. If a node appears at a location that is unoccupied on the left-hand side, a new node is created. Figure 12.1 shows an example rule: It is applied with a probability of $p = 0.3$ at every fourth time step ($\Delta t = 4$). One (arbitrarily chosen) node that fulfills *predicateX* and *predicateY* is affected by *actionJ* and *actionK*. Also note that a new node is created and is initialized in this rule for which no reference had existed before. In case there are at least 6 nodes that fulfill *predicateZ*, they will all be removed.

12.3.2 Algorithm

A swarm graph grammar $SGG = (\mathcal{I}, \Xi, \mathcal{G}_{predicate}, \mathcal{G}_{action}, P)$ is a quintuple, where $\mathcal{I}$ describes a set of individuals relying on rules and properties as explained in the previous section. At the beginning of the simulation, a set Ξ of axioms, in the form of initialization algorithms, is executed by (1) selecting and expressing individuals from $\mathcal{I}$, and (2) by assigning initial states to the newly created individuals. For a homogeneous swarm of nest-constructing wasps[1], for instance, $\mathcal{I}$ only has to comprise a single agent description. Having created a sufficient number of wasp agents, the axioms would assign contextual information such as an initial location to the individuals. In the main loop of the swarm graph grammar algorithm (Algorithm 1) two graphs $G_{predicate} \in \mathcal{G}_{predicate}$ and $G_{action} \in \mathcal{G}_{action}$ are subsequently created that merge the triggered predicates and corresponding actions of the individuals' local rules. $\mathcal{G}_{predicate}$ represents the set of possible graphs of individuals interconnected through predicates. $\mathcal{G}_{action}$ hosts all possible action graphs. Chains of relations among sets of swarm individuals create semantic topologies for global graph structures that describe the situational context or activity in the SGG system. Executing the actions of G_{action} yields the next simulation state after a policy P is applied to resolve possibly arising computational conflicts[2]. Thus, the alternating update of the graph instances $G_{predicate}$ and G_{action} based on the swarm individuals' behaviours drives the SGG simulation (Figure 12.2).

[1] See Section 12.4.2 for details.

[2] The implementation of an efficient conflict policy P is often difficult and its execution can be computationally expensive.

Algorithm 1 Swarm Graph Grammar: Main Loop

Require: $G_{predicate}$, optional: P
Ensure: alternating computation of $G_{predicate}$ and G_{action}
  repeat
    compute predicative graph $G_{predicate}$
    compute action graph G_{action} based on $G_{predicate}$
    apply order policy P to G_{action}
    execute ordered actions of G_{action}
  until simulation is terminated

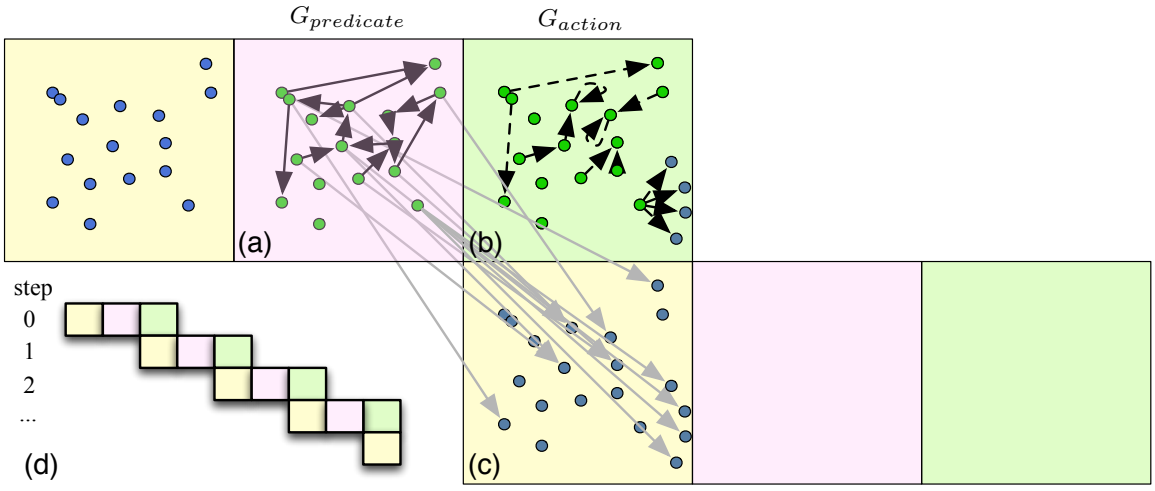
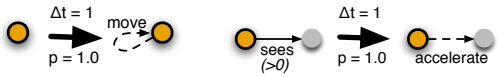


Figure 12.2: Subsequent computation of (a) $G_{predicate}$ and (b) G_{action} yield (c) the next simulation state. The grey arrows from (a) to (c) relate nodes to their contextual impact. (d) The simulation process is shown as a computation pipeline.



movement *acceleration*

Figure 12.3: Two rules to describe a boid agent's interaction behaviour.

12.4 Swarm Graph Grammars in Action

In this section we present three computational models realized with the SGG framework. We retrace (1) a simple boids simulation [1], (2) the stigmergic construction behaviour of the Chartergus wasp [2], and (3) cell proliferation induced by a set of growth factors.

12.4.1 Boids

In order to specify a standard boid flocking simulation [1], we use a swarm graph grammar $SGG_{boid} = (\mathcal{I}_{boid}, \Xi_{boid}, \mathcal{G}^{boid}_{predicate}, \mathcal{G}^{boid}_{action}, P_{boid})$. The sole individual $i_{boid} \in \mathcal{I}_{boid}$ contains several weights for flocking urges, parameters to determine a field of perception, as well as boundaries for the maximal flight acceleration max_{accel} and velocity max_{vel}. Ξ_{boid} generates a homogeneous set of swarm individuals that are initialized with a random position $\overrightarrow{p}$ and velocity $\overrightarrow{v}$. As no interaction conflicts arise, the policy P is empty.

Boids rely on two behavioural rules shown in Figure 12.3. The *movement* rule continuously updates a swarm individual's position in accordance with its velocity. The *acceleration* rule, substitutes the predicate $sees(u, v)$ with the action $accelerate(u, v)$. The predicate considers the reference node's location, orientation and perceptional field to select a set of interaction partners in accordance with their respective locations. The action also considers the difference between u's and v's states, including their locations and velocities, and accelerates u accordingly. For example, u accelerates towards v's location and it aligns its flight direction. In the example displayed in Figure 12.4, the boid agents form a cluster over time which is also reflected by increasingly connected interaction graphs.

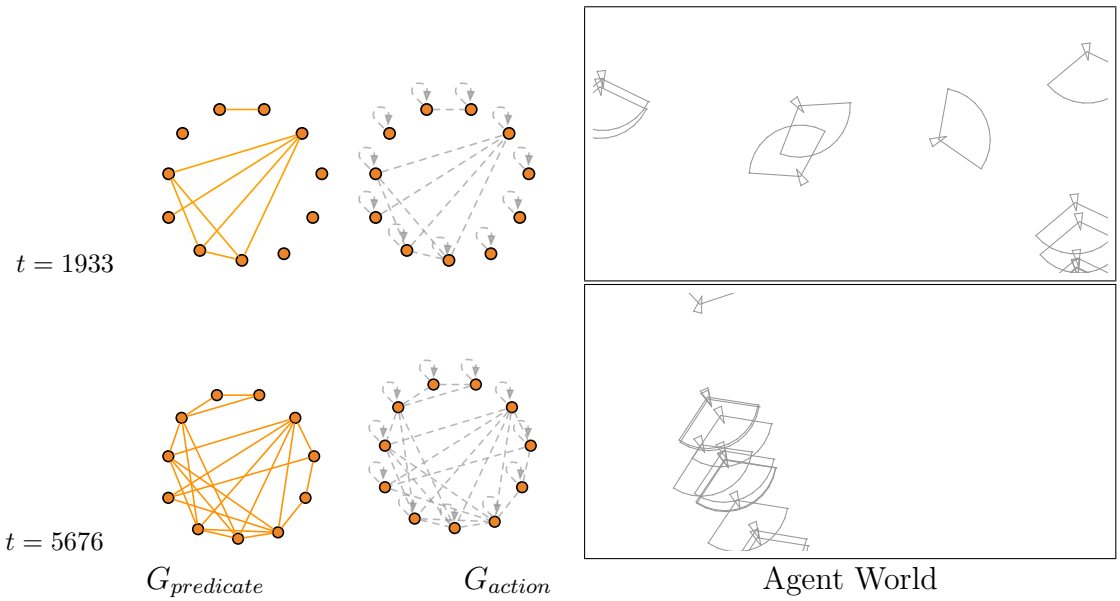


Figure 12.4: Two sets of graphs $G_{predicate}$, G_{action} and a visualization of the agent space show a clustering process in a SGG-driven boid simulation. The boid renderings—triangles oriented towards their velocity with a conic field of perception—partially overlap due to their strong alignment urge.

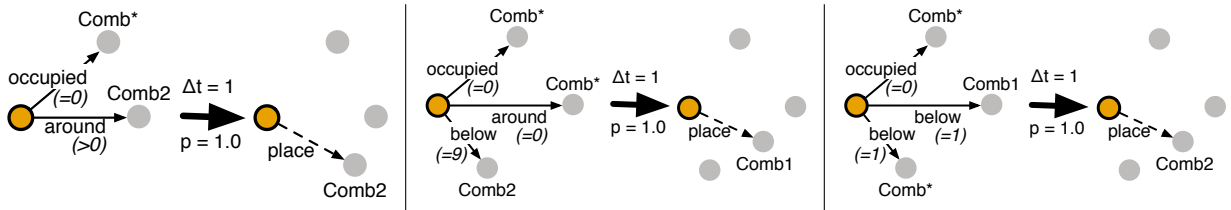
12.4.2 Stigmergic Construction

Theraulaz et al. have translated the nest construction processes of Chartergus wasps into individual behavioural rules [2]. The rules in Figure 12.5 closely retrace this behaviour[3]. The predicates *around*, *below* and *occupied* test the immediate surroundings of the wasp to trigger comb construction in the remaining rules. Hereby, previously deposited combs of two different types ($Comb1$, $Comb2$, or $Comb*$ for both) trigger the next *placement* actions. In addition, a *movement* rule as seen in Figure 12.3 moves an individual unconditionally to a random location in the simulation space. Figure 12.6 shows the development in agent space and correlates the activating (red) and the constructed combs (green). The rule deployment is shown in a series of interaction graphs $G_{interaction} = G_{predicate} \cup G_{action}$.

12.4.3 Swarm Development

Signalling factors determine the rate of cell proliferation which influence specific morphological developments [531]. The rules in Figure 12.7 configure cells which *grow* until they reach ma-

[3]The lattice-based matrix representation provided in [2] was translated into predicates that test the corresponding spatial relationships.



(a) floor extension (c) floor template (d) floor initiation

Figure 12.5: SGG rules that retrace the construction behaviour by the Chartergus wasp as described in [2].

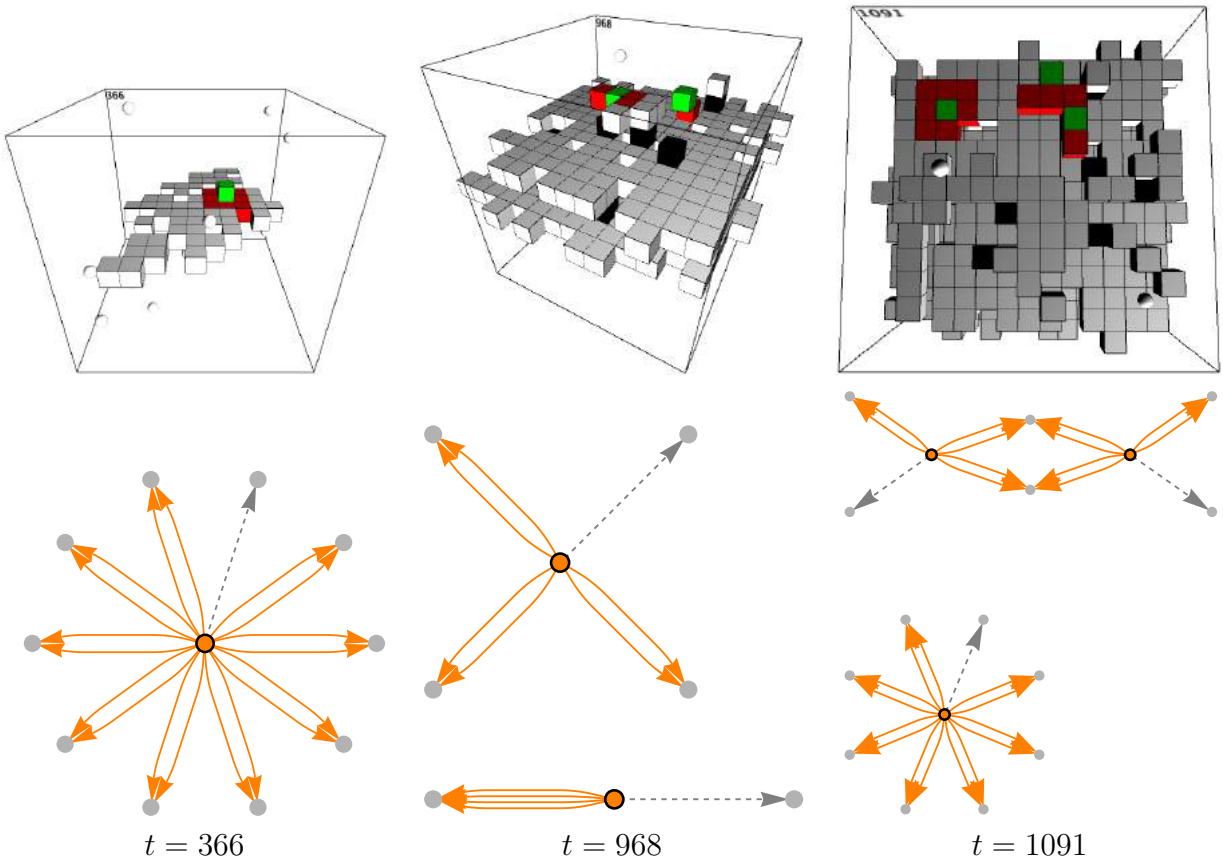


Figure 12.6: Agent space and the corresponding interaction graphs of a wasp-inspired construction process (grey dashed arrows indicate actions, orange ones predicates). At $t = 366$ a floor template is constructed (rule (c) in Fig. 12.5). At $t = 968$ the construction of a new floor is started (rule (d) in Fig. 12.5). At $t = 1091$ two floor extensions are performed by different wasp agents triggered by the same subset of combs.

turity (predicates *not mature* and *mature*). Mature cells that are *close to* a *Growth Factor* increase their internal *mitogen* concentration which in turn instigates proliferation (modelled as *reset* of the acting cell and *initialization* of a second cell).

Figure 12.8 shows screenshots of the simulation. Tissue cells within the vicinity of a signalling

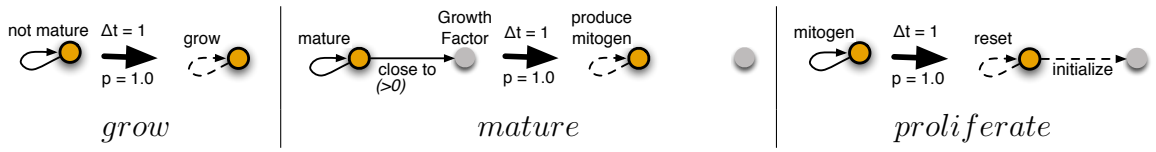


Figure 12.7: Three rules to describe a simple developmental process model.

molecule start proliferating. Collision resolution through an embedded physics engine allows the cells to assemble[4]. The emerging protuberance is slanted to the right in accordance to the initial distribution of signalling molecules. However, it is surprisingly symmetrical still, which could result from a lack of simulated cell polarization.

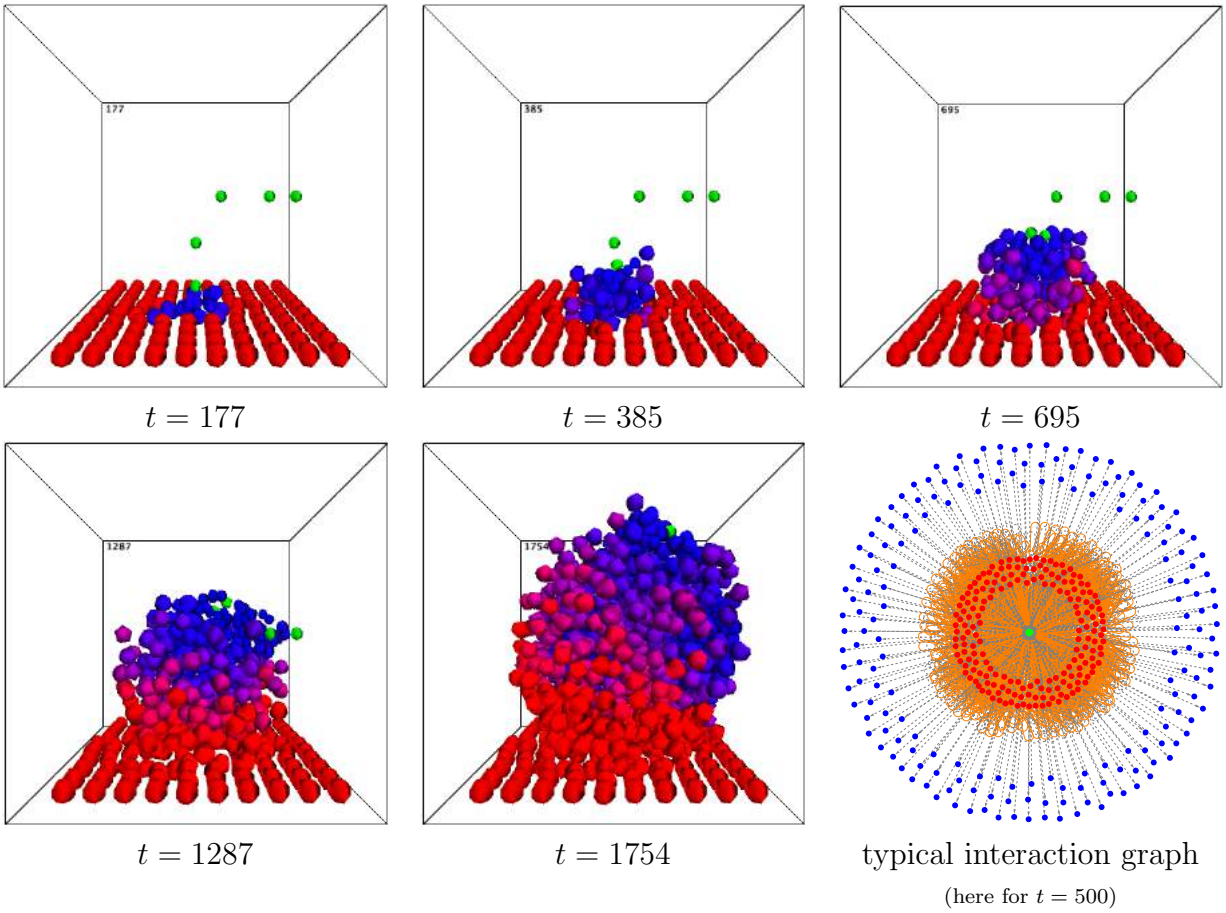


Figure 12.8: The proliferation of mature cells (blue: not mature; red: mature) is dependent on the proximity to growth factors (green). At any time of the simulation, large numbers of agents are informed by growth factors leading to typically dense but homogeneous interaction graphs.

[4]In the given experiment we rely on the Bullet physics engine, http://bulletphysics.org

12.5 Summary and Future Work

Swarm graph grammars are a modelling and simulation framework that provides a universal graph-based representation for swarm-based developmental systems. Besides metabolic operations, i.e. the creation or removal of agents, the semantics of agent relations are not part of the framework. The agents' abilities have to be implemented in the form of predicates and actions. The agents' rule sets (behaviours) drive the simulation processes and they are immediately reflected in the interaction graph of a simulation. As examples, we used SGGs to simulate boid flocking, stigmergic wasp nest construction, and growth and proliferation in cellular morphological processes.

We are currently working on several aspects to improve and harness the utilization of swarm graph grammars. The application of the framework has led to many refinements in respect to the formalism and the algorithm. However, in order to render modelling with SGGs accessible, especially to non computer scientists, we need to collect feedback from interdisciplinary modellers about the shortcomings of the representation, e.g. regarding its visualization, terminology and logic. In this paper, we have touched upon matching local agent rules with a simulation's emerging interaction graphs. We deem this a very promising approach to analyze emergent phenomena in simulations on the one hand, and to create complex interaction processes with dynamic interaction topologies on the other hand. Accordingly, systematic investigations have to be started. We are also working on a slight modification of the SGG framework so that nodes can encapsulate children and thereby computational or spatial hierarchies can be built. This would allow for hierarchical modelling as in P systems [523].

Acknowledgements

Support for this research was provided by the Undergraduate Medical Eduction program of the University of Calgary. We would like to thank Jörg Denzinger for his invaluable advice on multi-agent systems and Heather Jamniczky for her feedback on biological developmental systems.

Chapter 13

Modelling & Understanding the Human Body with Swarmscript

Sebastian von Mammen, Stefan Schellmoser, Christian Jacob, and Jörg Hähner. *Modelling & Understanding the Human Body with Swarmscript*. In: *The Digital Patient*, Wiley, 2016, pp. 149–170.

13.1 Introduction

It is well known that generic medical advice and therapy may have adverse effects on the patient's health [532, 533, 534] as well as on the health-care system as a whole, see e.g. [535]. The other way round, individualised treatment has been shown to be more effective, less invasive and reducing therapeutic side-effects [536, 537, 538]. 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 [88]. 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 [539] and means of interactive exploration of the physiological processes across the whole human body [540]. 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 [541], or magnetic resonance [542], 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 [543], predictions at the level of protein interactions [544], or proteome analysis [545]. 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 [121, 516, 515].

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 [16]. 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 [67]. Accordingly, we describe our approach in the light of various renderings immediately taken from SwarmScript runs. In the following section (Section 13.2), 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 13.3, we introduce the basic vocabulary and how to phrase sentences in the SwarmScript modelling language. This knowledge is put to the task in Section 13.4 where we demonstrate the application of SwarmScript by tracing a previously published agent-based model of the secondary human immune response [123]. In Section 13.5, 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.

13.2 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 [47, 546, 230]. 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.

13.2.1 Agents and their Representation

Agents representing the parts of a complex system have certain properties and behaviours [472]. 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 [55]. While this definition allows for arbitrarily complex blueprints of agents, our elaborations focus on reactive biological agents [547] 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 [548] has motivated efforts towards accessible agent-based M&S frameworks [61]. To some great extent, their designers have been quite aware of the need for understandable visualisation techniques [193] and accessible user interfaces [71]. 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 [549, 550]. In the 1990s, scientists from MIT developed a programmable LEGO brick that allowed to build robots from LEGO parts [198]. 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 [199]. 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 [111]. 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 [196]. 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 [551], StarLogo TNG [71], and Scratch [552].

13.2.2 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 [92]. 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 [553]. Hierarchical agent organisations are often used to express their spatial

arrangement, which is of great importance given the fact that interactions happen locally [114]. Hierarchical organisations are also a means to integrate multiple spatiotemporal model scales [554].

13.2.3 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 [184, 185]. 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 [186]. 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 [188]. 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 [555].

13.3 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 [540]. The standardisation accompanying the modelling process further allows to deploy automated, self-adaptive model optimisation routines [121]. 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.

13.3.1 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 13.2. 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.

Graph-based Rule Representation

Motivated by grammatical substitution rules that guide the interactions in developmental simulation models [3], we were initially pursuing a graph-based approach to the representation of behavioural rules [556]. As can be seen in Figure 13.1, 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.

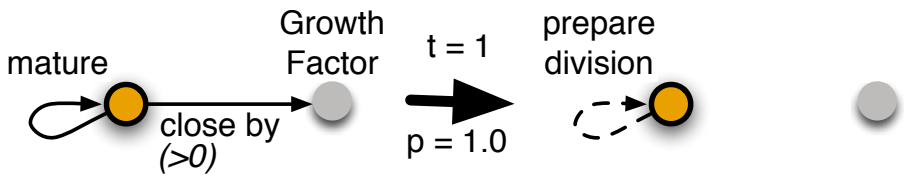
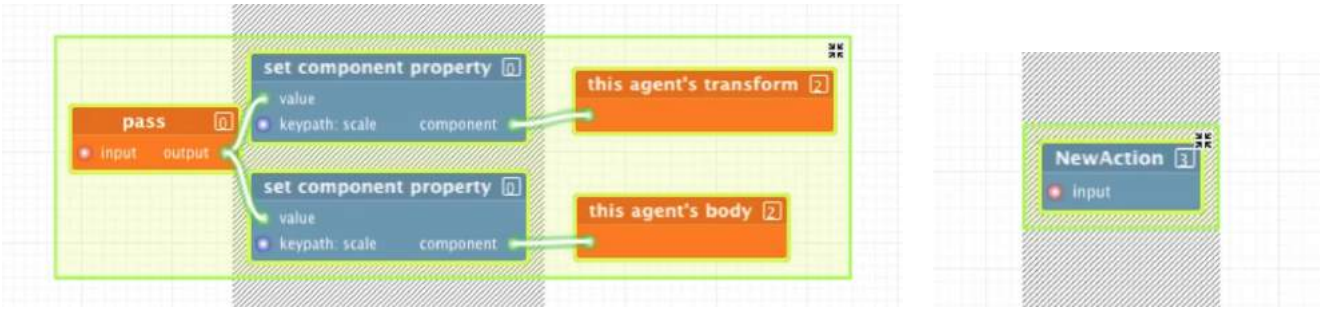


Figure 13.1: One of a set of graph-rewriting rules that describe the proliferation behaviour of a cell.

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 [92]. In particular, the next iteration of SwarmScript connected chains of operators (command objects without "needing any detailed knowledge of how the rest of the program works" [557]) 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 13.2 (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 [558] (see Figure 13.2(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).

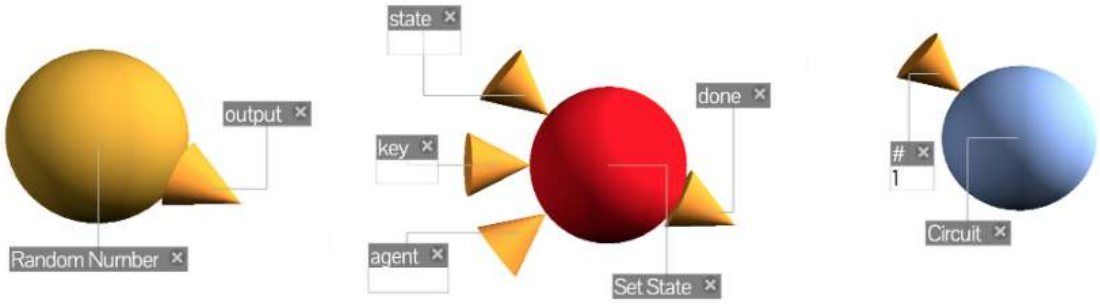


(a) (b)

Figure 13.2: 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.

13.3.2 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 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 13.3 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.



(a) (b) (c)

Figure 13.3: 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.

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 13.4 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.

13.4 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 [304, 384, 123]. 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.

The domain model itself stages several types of agents (Figure 13.5) that react based on spatial collisions and internal states. Epithelial cells (Fig. 13.5(a)) that constitute lung tissue are susceptible to infection by Influenza A viruses (Fig. 13.5(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 [123], a comprehensive set of

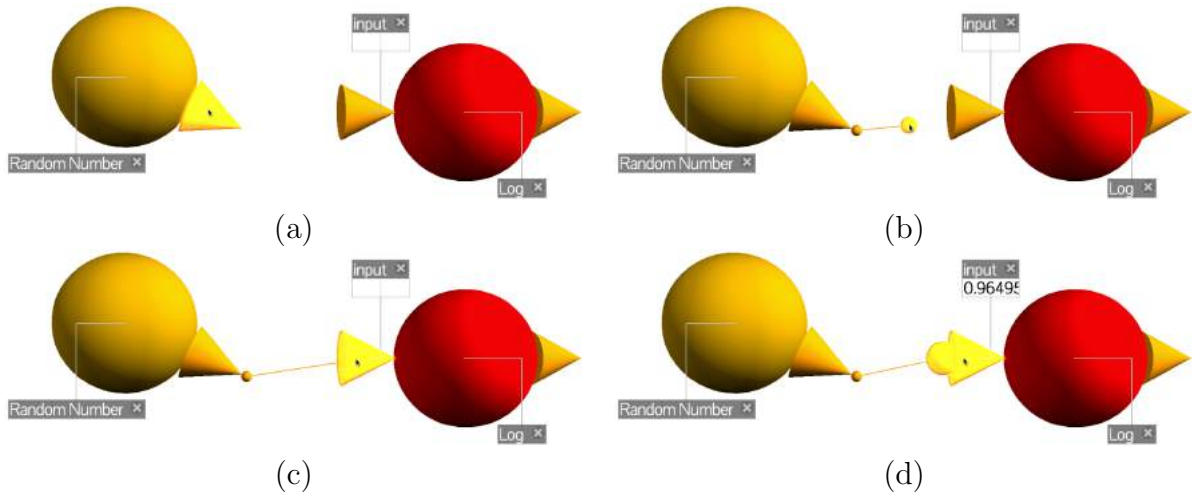
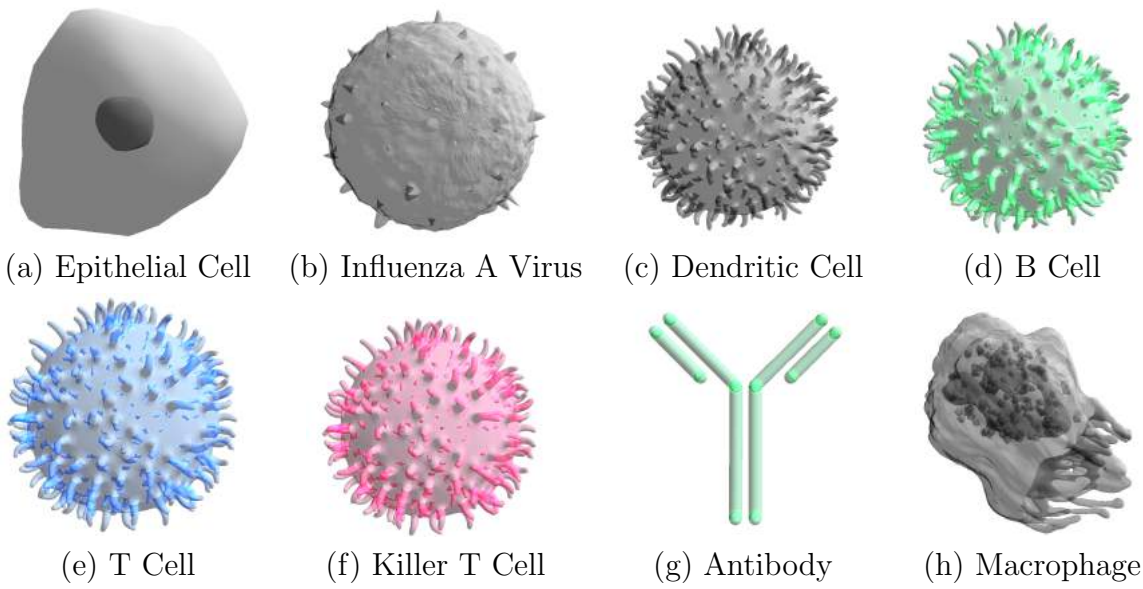


Figure 13.4: (a) Dragging across an output connector creates a new edge, (b) whose head is navigated by the user, (c) to be dropped onto an input connector of another operator. (d) The new connection has established a properly phrased behavioural rule. Therefore, the action operator now receives and processes input information as seen in the input connector's label window.

behavioural constants is presented that reliably retrace the progression of the immune response. After exposure to the viruses, dendritic cells (Fig. 13.5(c)) that are scarcely spread across the lung tissue, migrate to the lymphatic system to activate B (Fig. 13.5(d)) and T cells (Fig. 13.5(e)). Some of those mature into cytoxic killer T cells (Fig. 13.5(f)) and destroy infected epithelial cells, terminating the viral spread. B cells boost the production of antibodies (Fig. 13.5(g)) that attach to the viruses – such opsonised viruses are then destroyed by macrophages (Fig. 13.5(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 [123], 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 13.6).

Each visual object that represents a biological agent (Figure 13.5) is augmented with the according SwarmScript behaviour as seen in Figure 13.7. Floating labels provide clarity about

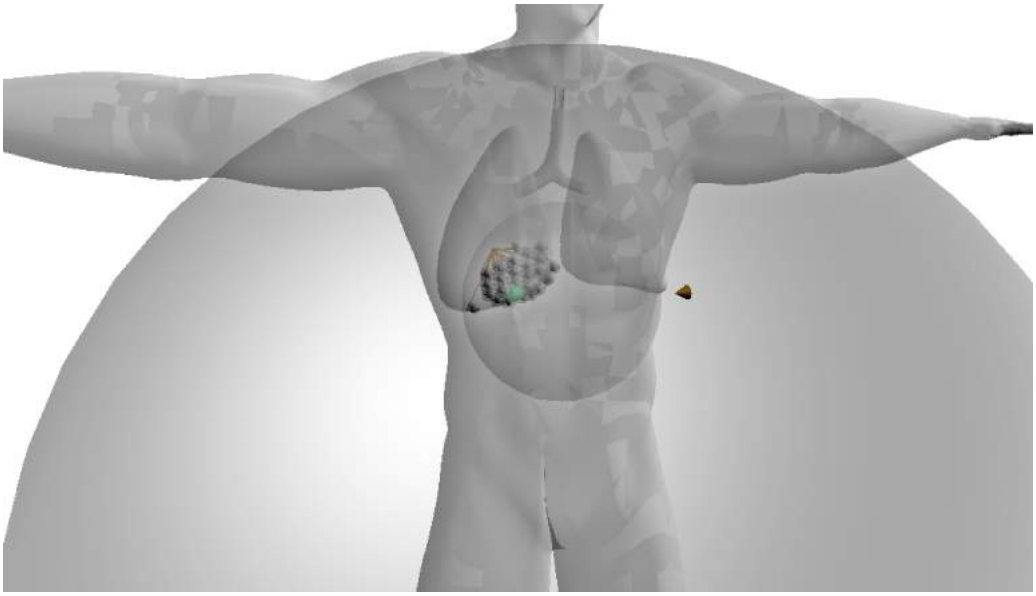


(a) Epithelial Cell (b) Influenza A Virus (c) Dendritic Cell (d) B Cell

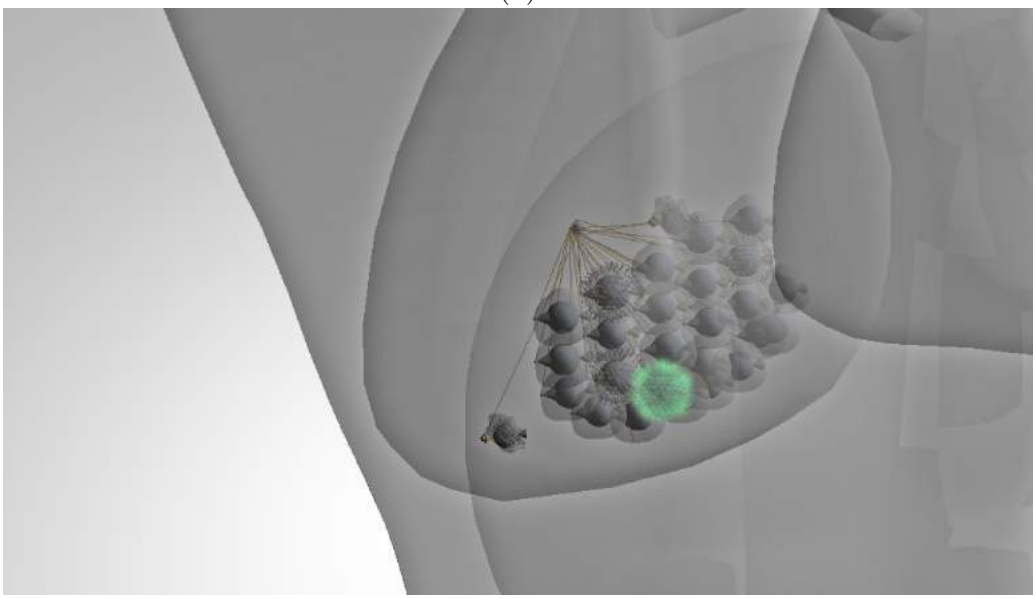
(e) T Cell (f) Killer T Cell (g) Antibody (h) Macrophage

Figure 13.5: Visualisations of the biological agents that drive the SwarmScript INTO3D simulation of the secondary human immune response to Influenza A.

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 13.4. 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 13.7). Based on a set of primitive query and action operators, 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 13.7(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 13.9 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.

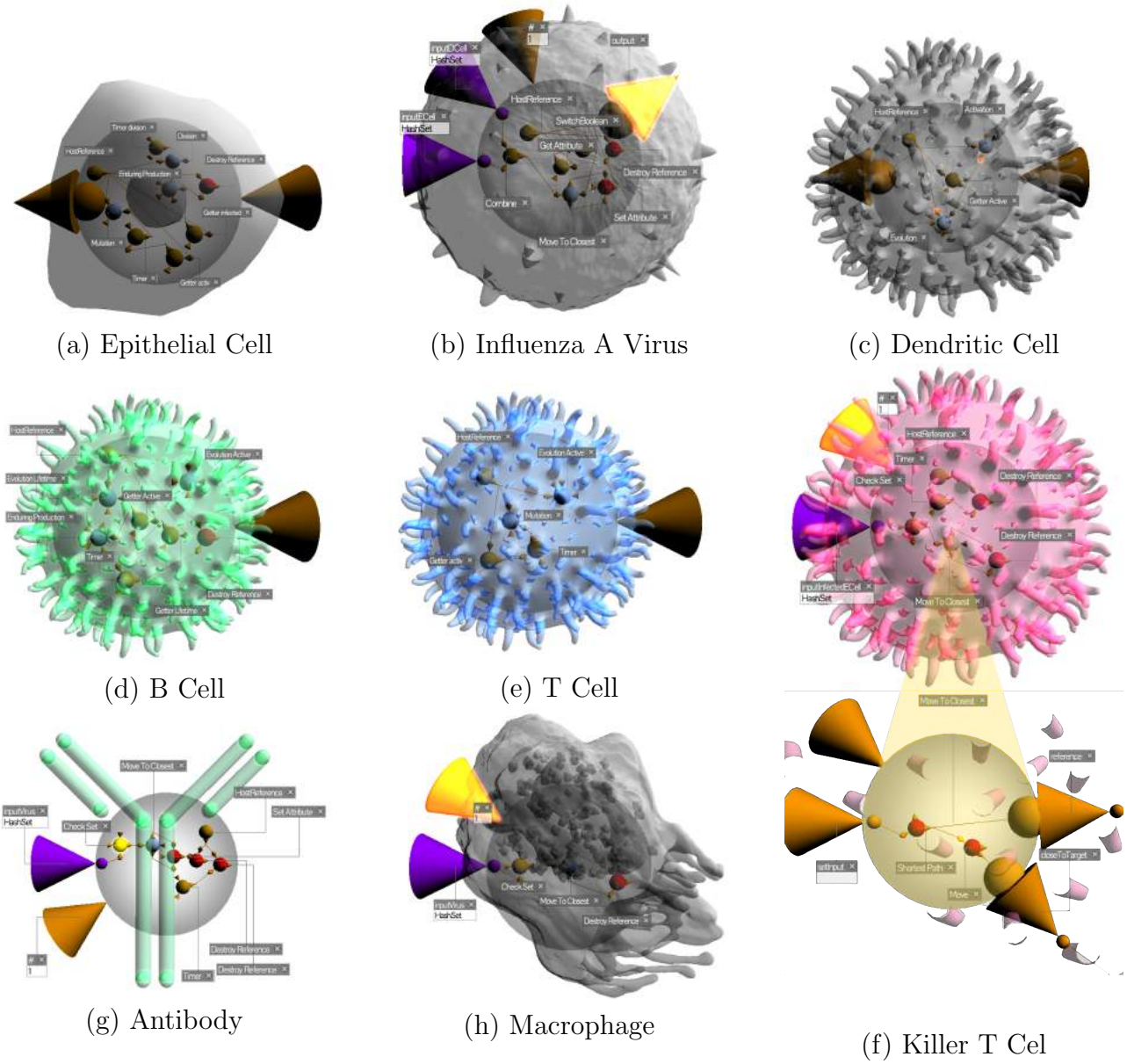


(a)



(b)

Figure 13.6: 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

(g) Antibody (h) Macrophage (f) Killer T Cel

Figure 13.7: 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).

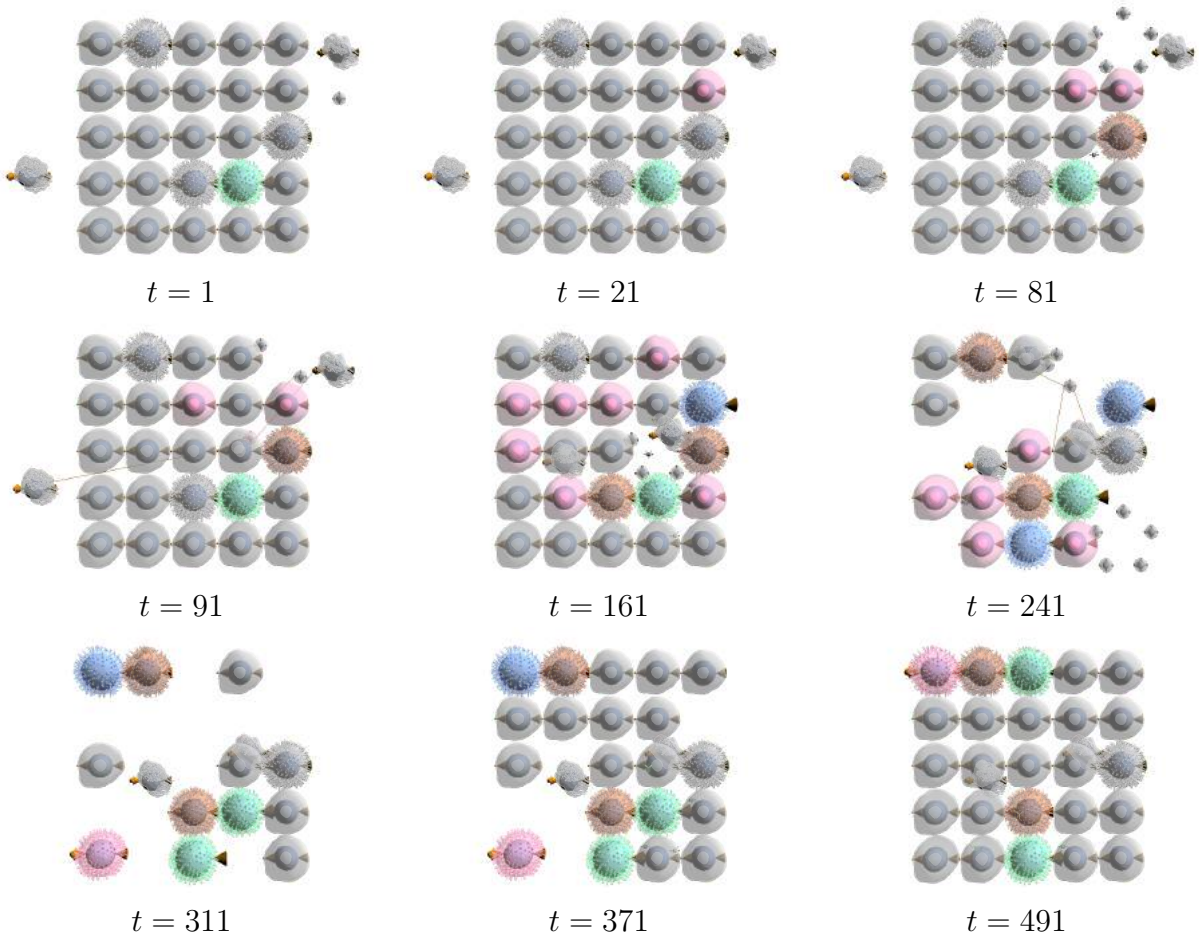


Figure 13.8: The secondary immune response simulation initially hosting twenty-five tissue cells at different time steps t. The progression shows the initial infection, the reaction of the macrophages, the differentiation and recruitment of lymphocytes, the viral spread, the production of antibodies, 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.

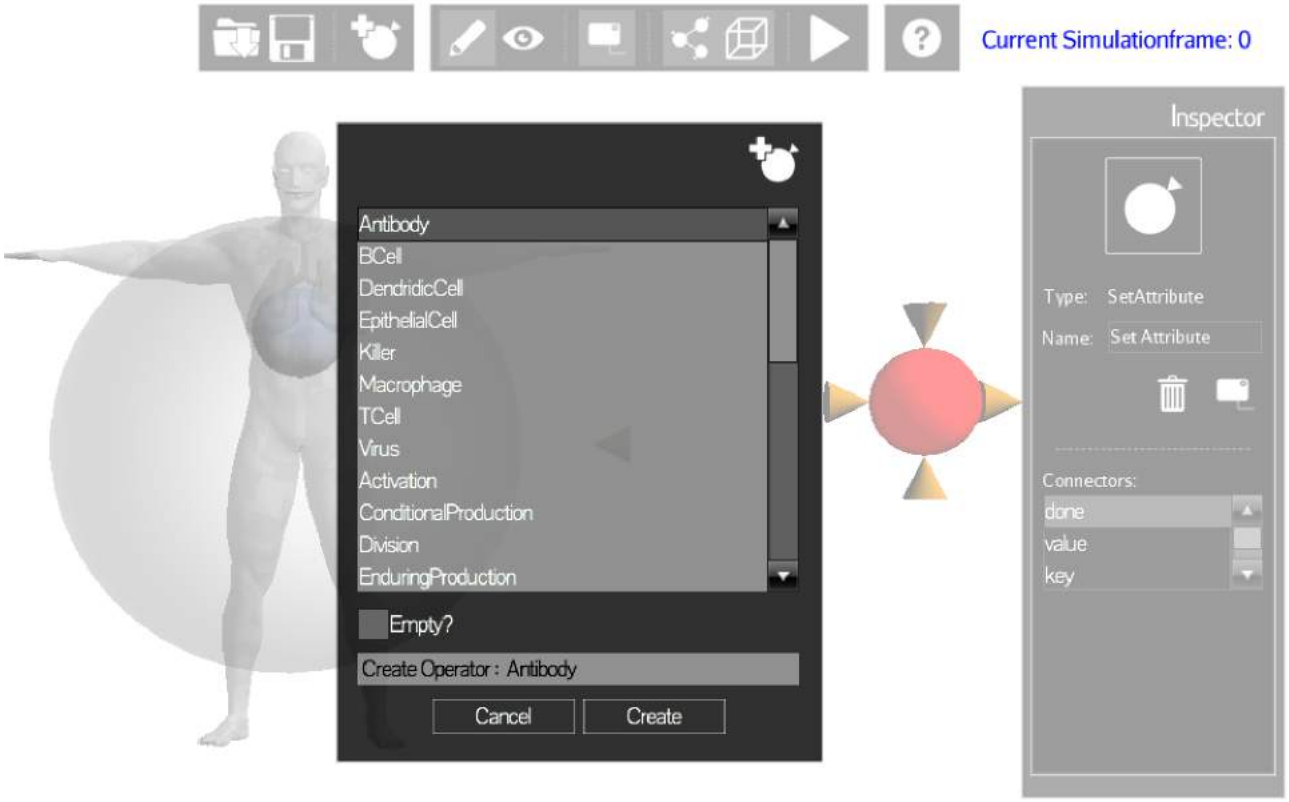


Figure 13.9: The heads-up display for navigating the modelling & simulation phases with SwarmScript INTO3D (top), for selecting and deploying previously stored operators in the current scene (centre), and to configure the currently selected operator (right).

13.5 Discussion

From its graph-based predecessors [556], 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.

13.5.1 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 [114] and [547]. 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 [196]. 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 [559]. In SwarmScript, all variables that are queried or modified can be modelled explicitly. For instance, the operator $HostReference$ (e.g. in Figure 13.7(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 13.7, 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.

13.5.2 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 [501], 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 [20]. 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 [560] combined with novel technologies [561]), 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 [562]. 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 [563]. The combination of heads-up displays for navigation and introspection as seen in Figure 13.9 already addresses the need for hybrid display modalities to a very limited extent. Similar to the rich options provided by visual analytics [539], increasing the number of modelling modalities might prove useful in the context of SwarmScript-based models.

13.6 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. j

Chapter 14

Swarm Grammars GD: Interactive Exploration of Swarm Dynamics and Structural Development

We present an interactive simulation of Swarm Grammars (SGs). SGs are an extension of L-Systems, where symbols of the production system are considered agents, whereas the given production rules determine their differentiation or reproduction. Assigning boid properties to the SG agents yields spatial dynamics apt to building structures in space and to collaborate stigmergically. In the presented interactive simulation, we put an emphasis on accessible interactive visuals for shaping the initial configuration of the simulation, to program the agents' perceptual and productive behavioural abilities, to dynamically drive developmental stages and to fine-tune visual structural properties such as colouring and scaling of the utilised developmental building blocks. Our system has been successfully deployed to promote swarm dynamics and developmental processes as important aspects of Artificial Life in a playful way. We present results from deploying the simulation in the context of an event to promote STEM research among high-school girls.

Sebastian von Mammen, Sarah Edenhofer. *Swarm Grammars GD: Interactive Exploration of Swarm Dynamics and Structural Development.* In: *Proceedings of ALIFE 14: The International Conference on the Synthesis and Simulation of Living Systems*, MIT press, 2014, pp. 312–320.

14.1 Introduction

Confronted with the challenge of providing an engaging entrance point to Artificial Life research to young high-school students, we decided on implementing an interactive Swarm Grammar (SG) simulation [3]. SGs seemed to be an adequate choice, as they integrate aspects of complex system dynamics—bridging from the behaviour or simple individuals to the emergent properties of large populations—and developmental processes—yielding unique artefacts that allow to trace spatial interaction processes over time.

In order to kindle intrinsically motivated engagement [427], the simulation was devised to (a) establish a relationship between the students, the software and its artefacts. (b) We had to provide accessible means to defining rules and configuring agents so the students could challenge their competence in the actual simulation processes and explore the outcome of their high-level programming ingenuity. (c) The intensity of the students' explorations and design efforts has to result from their voluntary engagement. Accordingly, the simulation provides an open-ended, directly manipulatable playground environment that is freely and widely accessible as a public, WebGL-driven website[1].

In the remainder of this paper, we present the following aspects of the project. In the next section, we briefly summarise the key concepts from related works that we utilised in the agents' flocking definition and their (re-)production rules. Afterwards, we explain the simulation concept with an emphasis on its visual programming assets. Before concluding this work with a brief summary and an outlook on possible future work, we dedicate one section to presenting select simulation artefacts as well as preliminary user feedback.

14.2 Related Work

Our simulation concept has been developed to support the definition of perception and actuation properties of 'boid' agents and the construction, reproduction or differentiation of swarm grammar agents. The deployed mechanics of user interaction and visual programming techniques that will be outlined in the next section, are a translation of the following, underlying

[1] `http://www.vonmammen.org/SG-GD/`

geometrical and grammatical relationships.

14.2.1 Boid Flocking

[1] presented the concept of 'bird-oids', or boids. It implements smooth, decentralised steering of swarms of agents based on several 'flocking urges' that emerge from the relationships between an agent and its neighbours. The neighbourhood is determined by the agent's radial field of view (FOV) defined with a maximal distance d_{max} and an angle α, see the annotated screenshot from our simulation in Figure 14.1.

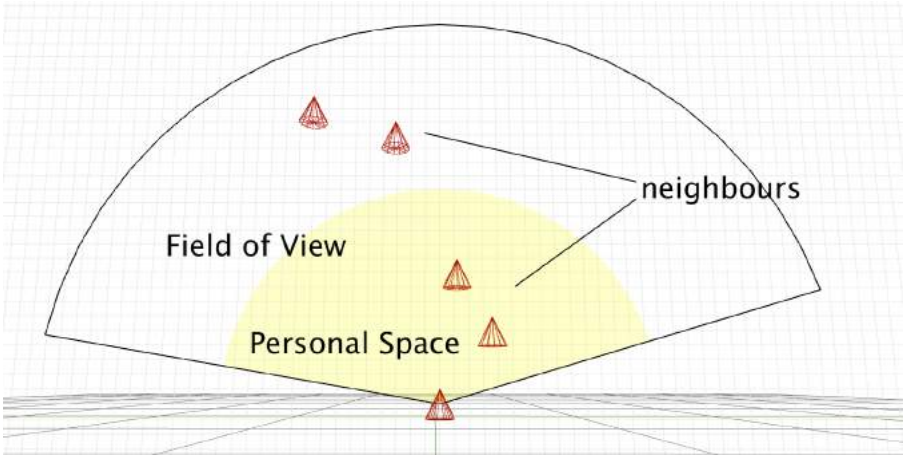


Figure 14.1: Perception of a 'boid' agent. Agents within its field of view are perceived as neighbours, those within the 'Personal Space' are considered too close, triggering an evasive manoeuvre.

Neighbours within a minimal distance d_{min} are considered 'too close' and trigger an evasive manoeuvre away from their centre (*separation urge*). The agent further synchronises its velocity (heading and speed) with the other neighbours (*alignment urge*) and it accelerates towards their centre (*cohesion urge*). Coefficients of these acceleration urges determine the emergent flocking behaviour. In addition to *separation*, *alignment* and *cohesion*, we utilised a correspondingly weighted random vector (*random urge*) and a global direction vector (*direction urge*) to steer the individual agent. The acceleration $\vec{a}_i$ of an individual i totals its j neighbour-dependent urges $\vec{u}_{ij}$, scaled by the individual's weights w_{ij}. The agents' acceleration and flight are kept within reasonable boundaries by maximal values for acceleration, a_{max}, and velocity, v_{max}. For simplicity sake, we chose integration step size $\Delta t = 1$ to infer the updated position p'_i of individual i. The corresponding equation system is listed below; markings denote the sequence

of variable updates, $|\vec{x}|$ denotes the norm and $\hat{\vec{x}}$ the versor of vector $\vec{x}$.

$$\begin{aligned}
\vec{a}_i &= \sum_j w_{ij}\vec{u}_{ij} \\
\vec{a}'_i &= max(a_{max}, |\vec{a}_i|)\hat{\vec{a}}_i \\
\vec{v}'_i &= \vec{v}_i + \vec{a}_i\Delta t \\
\vec{v}''_i &= max(v_{max}, |\vec{v}'_i|)\hat{\vec{v}}'_i \\
\vec{p}'_i &= p_i + \vec{v}''_i\Delta t
\end{aligned}$$

In Figure 14.2, all flocking parameters are shown that the user is encouraged to alter. Similarly to Reynolds' later extensions [564], we give the user the ability to introduce novel agents, to change their FOV and flocking weights on the fly and, thus, to interactively guide the emerging flocks.

Figure 14.2: Boid parameters as displayed to the user and offered for alteration. Changes to the Field of View parameters are immediately reflected by the neighbourhood visualisation of the introspected agent.

14.2.2 Swarm Grammar (Re-)Production

Originally, a swarm grammar $SG = (SL, \Delta)$ was conceived as a combination of a rewrite system $SL = (\alpha, P)$ and a set of agent specifications $\Delta = \{\Delta_{a_1}, \Delta_{a_2}, ...\Delta_{a_n}\}$ for n types of agents a_i [3]. The rewrite system SL loosely followed the concept of an L-system with axiom α and production rules P, as described by [508]. In the simplest form of context-free 0L-systems, each rule has the form $p \rightarrow s$, where $p \in \Omega$ is a single symbol over an alphabet Ω, and $s \in \Omega^*$ is a word over Ω. The application of the replacement rule can be conditional, for instance upon a successful stochastic experiment (with specified probability θ) or repeatedly over time (with a specified time period Δt).

Agent specifications may include the flocking parameters described above such as the agents' FOVs and urge weights, as well as characteristic parameters of the geometrical objects they leave behind during their flight. Figure 14.3(a) shows a representative artefact produced by an early swarm grammar, emphasising the branching structure emerging from the reproduction rules $SL = \{A, \{A \rightarrow BBB, B \rightarrow A\}\}$ in combination with a moderate separation urge.

(a) (b)

Figure 14.3: (a) An early SG definition emphasising branching [3]. (b) Artefact built by an extended, stigmergic SG [4].

Later, the rule representation of Swarm Grammars was extended towards quantitative stigmergy [345], which (a) allowed to trigger (re-)production, in case a specific environment is perceived and (b) to utilise various static building blocks as well as agent progeny as product. Hence, spatial structures became the outcome of the agents' behavioural interactions rather than simply tracking their flight. A structure built by an extended SG is shown in Figure 14.3(b).

14.3 Interactive Simulation Concept

Our application offers several user interaction mechanisms that support the population and configuration of the simulation space. In order to keep the user interface free from clutter, we decided to omit certain functionalities altogether, such as the rotation of geometric bodies, and we stripped the UI of any data that would not immediate benefit the target audience, such as the bodies' coordinates. At the backend, too, we pursued a simple but still ambitious management of agent specifications and geometric templates.

14.3.1 Basic Scene Manipulation

Figure 14.4 shows a close-up of the top-left corner of the main screen. Here, the user can start and stop the simulation. In the pull-down 'templates' menu an agent specification or a static geometry can be chosen to populate the simulation space. Clicking on any object in the scene (initially, there is the ground) places the selected template on top—in this way, the user can stack objects and populate in all three dimensions (Figure 14.5). Click and drag of an object moves it parallel to the ground. Hovering above an object and pressing the minus key removes an object (we found that the delete or backspace key is frequently assigned to other tasks in standard internet browsers).

Right-clicking an object exposes its properties, as seen in Figure 14.5, and allows the user to change them. Property changes apply to all objects of the same name, which is shown as the top-most entry in the introspection menu to the right. An alteration of a name triggers the creation of an according, new template in the 'templates' drop-down menu (Figure 14.4). In this way, the user can create a diverse set of static geometric objects and agent specifications.

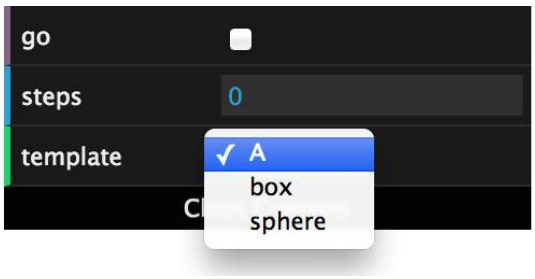


Figure 14.4: The menu of the simulation's main screen. The simulation process can be started, feedback about the current simulation step is provided and a template can be selected to populate the simulation scene.

Figure 14.5: Templates are placed on top of any clicked, existing objects. Properties of objects in the scene can be introspected and change on right click; an according menu appears in the upper-right corner of the screen.

Boid flocking parameters, including the field of view, and the (re-)production rules are part of an agent specification. As we offer a visual programming interface to configure some of these properties, the camera positions itself at a predefined distance from the agent when introspected. The dolly animation closing in on an introspected agent is shown in Figure 14.6.

14.3.2 Rule Editor

During introspection of an agent specification, alterations of the field of view parameters, d_{min}, d_{max}, α, result in an updated visual representation. This immediate reflection helps the user relate the parameter values to actual geometric dimensions and to quickly grasp the variables' relationships. Yet, the visualisation of the field of view plays another, more important role.

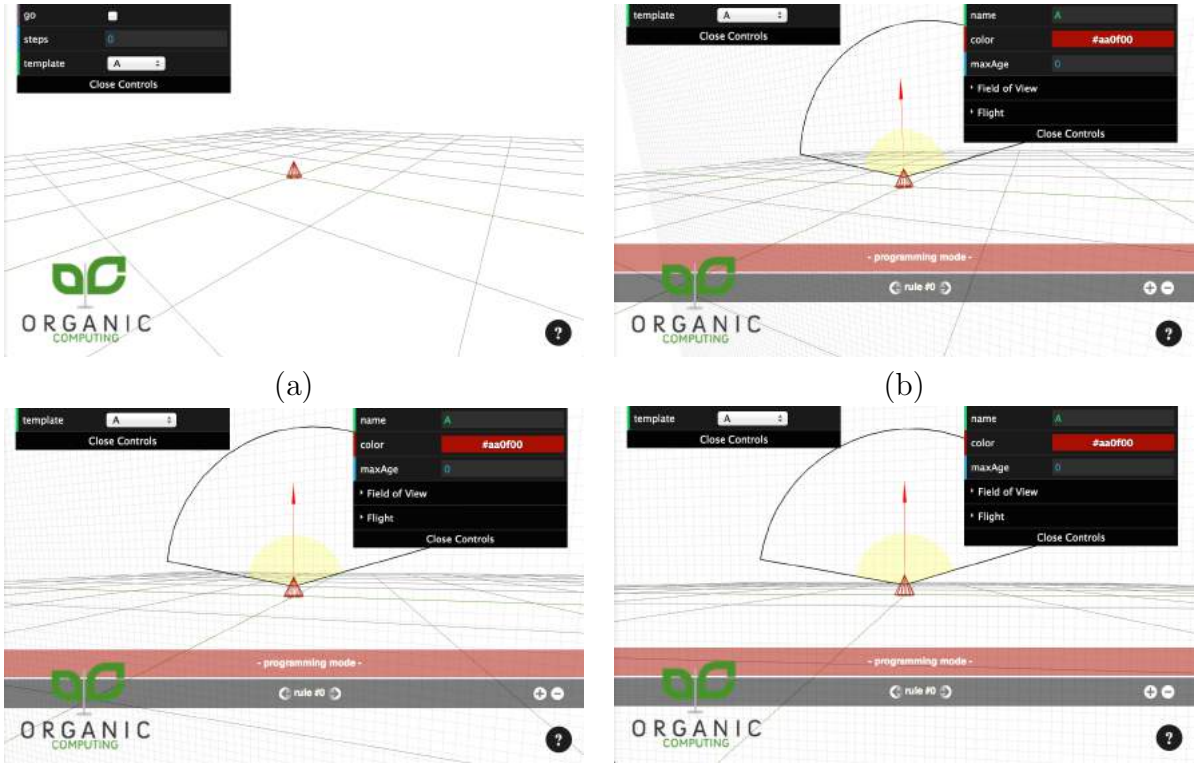


Figure 14.6: (a) A Swarm Grammar agent placed on the ground. When right-clicked, the camera automatically positions itself at a predefined distance (b-d).

Figure 14.7 shows the view of an introspected agent. Production rules can be specified directly in the vicinity of the agent. In particular, dice, timers, and and arrow-enclosed timers can be dropped, which represent the abstract rule conditions and associate probability θ, point in time t, and time interval Δt variables, respectively. Platonic solids and agents that are placed outside of the introspected agent's FOV are products of a behavioural rule, those inside are considered quantitative, stigmergic conditions. Any such stigmergic conditions are fulfilled, if the according number of objects is perceived by the agent in the neighbourhood or in the personal space, respectively. For production objects, the minute geometric offset from the introspected agent is taken into account. Their displacement as expressed visually in the rule determines their relative placement in the simulation, which resolves the issue of potentially conflicting product placements.

As a consequence of the semantics associated with differently located platonic solids and agents, their placement and movement are diligently tracked and registered. The red bar towards the bottom of the screen provides feedback about the user's programming efforts and any corresponding rule-affecting changes. The grey bar below provides the user with information about the currently displayed rule, to create new rules, to remove existing ones and to browse the complete set of rules of the introspected agent (and its namesakes).

14.4 Preliminary Feedback & Example Outcomes

In this section, we introduce the circumstances of the first deployment of the interactive simulation presented in this paper, *Swarm Grammars GD*. We provide details on the preliminary feedback we have gathered and we give examples of the artefacts built in this context.

14.4.1 Girls-in- STEM Programme

The concept was conceived while developing contributions to an entertaining and informative programme that aims at encouraging girls to develop and follow their passion for Science, Technology, Engineering and Mathematics (STEM) at the Faculty of Applied Computer Science at the University of Augsburg, Germany. As part of this programme, four groups of roughly ten girls between the ages twelve to fifteen attend four slots of 45 minutes each, offering different

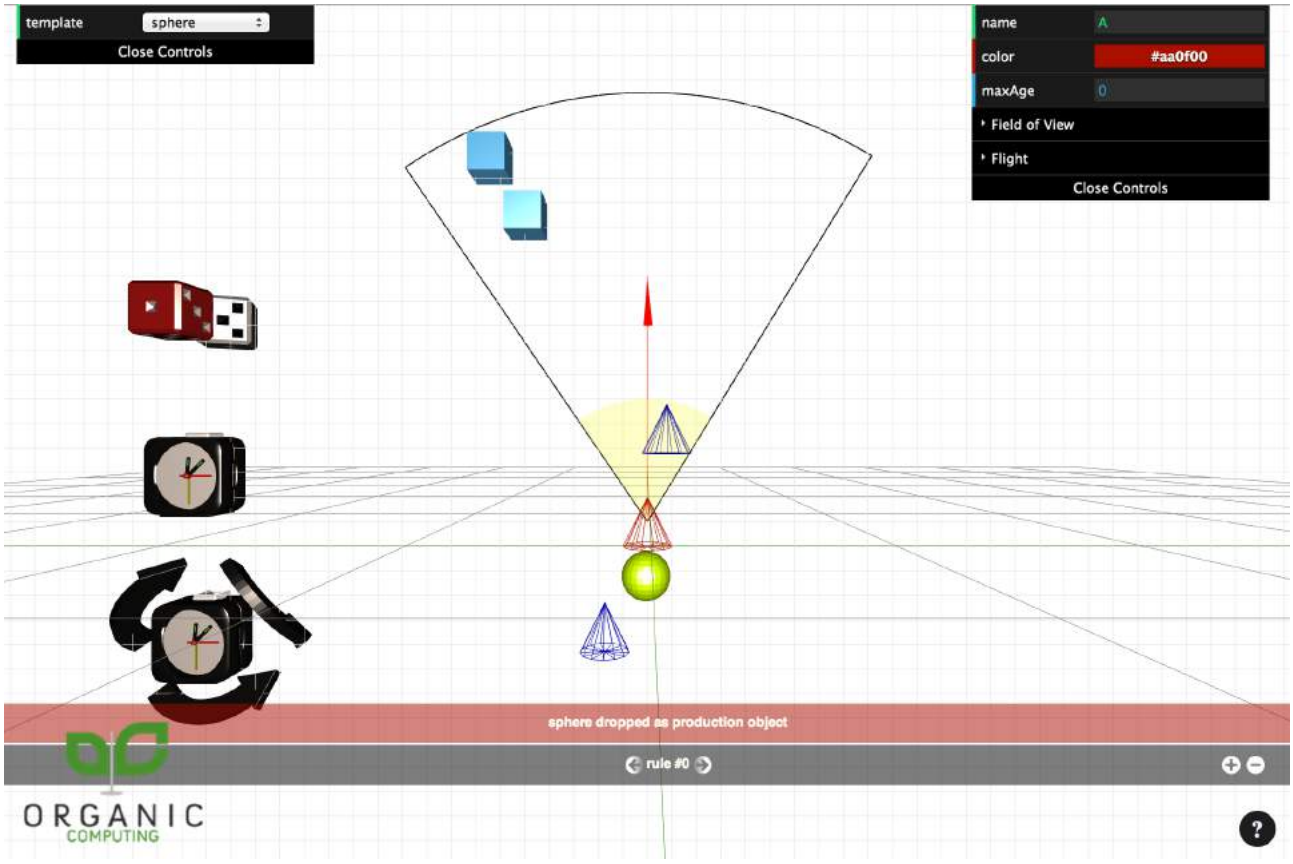


Figure 14.7: Introspecting an agent specification, sets of production rules may be visually programmed. Abstract conditions, stigmergic conditions, and products of the rules can be placed in the local vicinity of the agents.

contents and activities: 'Autonomous Vehicles', 'Sight- Finder', 'Touch-Robots', and our entry ' Artificial Life'.

14.4.2 Aspects of Artificial Life Research

We identified the following aspects of Artificial Life research that our implementation makes accessible to interested novices in a playful manner.

Similarly to popular simulation environments such as NetLogo [565], *Swarm Grammars GD* promotes an agent-based modelling approach. More specifically, it provides direct access, often supported by visual cues and interactive elements, to the parametric properties and sets of "if-then"-rules that describe the behaviour of deterministically or stochastically acting, spatially interacting reactive agents [472]. When occurring in greater numbers, the interaction of such agents may result in complex feedback cycles, which in turn might lead to emergent phenomena,

such as flocking dynamics [117] or complex built constructions [229]. The means to directly program an individual agent or to simultaneously modify all agents of a certain type allows one to observe and experiment with the relationship between local behaviours and such global emergent patterns, as for instance portrayed by [298].

As described in the section on Related Work, the interaction mechanisms that individual agents can perform in *Swarm Grammars GD* are limited to neighbourhood-dependent boid flocking [1] and rule-based production as in advanced swarm grammar concepts [3]. Both can be considered concrete concepts of two important Artificial Life themes, namely *collective locomotion* and *developmental models*. We make the first theme accessible by offering the means to visually program an agent's perceptional abilities. It is further promoted as simple construction rules that merely place single three-dimensional objects behind the agents at each simulated step effectively trace the resulting flight dynamics, as for instance seen in Figure 14.8(a).

Regarding the latter theme, *developmental models*, *Swarm Grammars GD* touches on the aspects of production, reproduction and differentiation, whereas these processes are triggered by the agents' internal states, effected by timers and stochastic experiments, as well as external stimuli (see our explanations of the visual rule editor above). Motivating differentiation based on locally perceived stimuli, such as the presence of a specific construction template or of a peer of a specific type, enables modelling a mechanism similar to task assignment in social insect societies [230]. Stimulus-dependent construction efforts, on the other hand, allow one to implement sigmergic lines of communication [566], i.e. indirect communication through the environment. All elements in *Swarm Grammars GD*, whether they are agents or built objects, have a lifespan attribute which determines the respective element's timely removal from the simulation (the elements are not removed, if this attribute is set to the value 0). Utilising such a timed appearance in combination with stigmergic construction and boid-based cohesion, a simplistic model of Ant Colony Optimisation can be retraced [567].

14.4.3 Presentation Sequence

Despite the intricate modelling options *Swarm Grammars GD* provide the user with, in the context of a short introduction to young novices, we directed our introductory STEM session to Artificial Life to the foundations of agent-based programming and discussed the intuitively

accessible basics of swarm dynamics, construction and reproduction. After brief examples of L-Systems [508] and boids [1] and their utilisation as special effect techniques by the movie industry (schematic slides and movie snippets), we introduced *Swarm Grammars SG* hands-on. Following the structure of the above section 'Interactive Simulation Concept', we first explained the main view of the simulation space and basic user interactions to populate it. Next, we briefly demonstrated the exploration potential merely arising from altering various boid parameters. Finally, we detailed the composition of production rules (in this order: producing static geometries, initialising other agent specifications, and adding conditions to the rules). At this point, each group had approximately 25 to 30 minutes at its disposal for exploring and design artificial artefacts and swarm dynamics, under guidance and with feedback if desired. Our offer to print out screenshots of the individually generated artefacts was in good demand, we handed out 24 of them.

14.4.4 Leeway for Improvement

Some weaknesses of the current simulation became obvious during the supervised sessions—especially with respect to choosing reasonable parameter values, including boid urge weights, and configuring abstract production rule conditions such as chance or time steps. One could mitigate the issue of conflicting or ineffective boid parameter sets by offering several presets such as the ones evolved by [376]. Regarding conditional values, if one does not want to drastically limit the expressiveness of rule compositions, warnings could be issued that hint at potentially unreasonable parameters. For instance, agent multiplication at high frequencies quickly exhausts the host computer, if the maximal life span of the agents is relatively high.

While exploring, one student asked how one could program the cubes (as opposed to the agents) to reproduce themselves. This question made clear that the distinction between producing agents and static geometries is arbitrary, not necessary, and possibly not even beneficial for the sake of functional distinction. At least one could expand the computational representation and nullify this rigid distinction. Other, more frequently asked questions were related to increasing the diversity of templates: although creating new templates by entering new names was quickly understood, this mechanism seemed to be too lengthy for supporting the creative diversity in colour and scale some users would have liked to deploy.

Despite the shortcomings of the current implementation, twelve out of a total of 42 participants declared our session and the use of *Swarm Grammars GD* to be the highlight of the whole introductory programme.

14.4.5 Example Artefacts

Figure 14.8 shows an array of six different Swarm Grammar artefacts programmed by participants of our session. We can identify different classes of structural complexity based on the flocking and production rule complexities of the SG agents. With only minor changes to the default boid flocking parameters (see Figure 14.2) and continuously dropping geometries, swarm motion is captured by the built artefacts (Figures 14.8 (a-c)). Agents with distinct construction behaviours—either resulting from differentiated reproduction or from interactively adjusting agent specifications—yield more visually complex structures (Figures 14.8(d-f)).

14.4.6 Perspective Shots

Unfortunately, as the students did not have much time to explore, play and create, their artefacts were mostly captured from a global perspective, which usually does not emphasise their appealing peculiarities. Figure 14.9 displays several Swarm Grammar configurations, snapshots of the emergent generative processes and close-ups of the final products set in scene.

In particular, three different Swarm Grammars are shown. The first one, with captions subtitled *cloud*, works based on a simple, unconditional production rule that traces an upwards-flocking agent with a cluster of spheres. The twists and turns triggered by the interplay of several flocking agents (as seen in Figure 14.9(b)) are exalted in the close-up by an upwards perspective and light shading. The second example deploys two agents, the first one simply flies upwards, repeatedly creating an offspring of a different kind (rule^0_{wall}). The latter one is pushing hard to the right while continuously dropping cubic solids (rule^1_{wall}) but it is also distracted by its neighbours and thrown off its path by some randomness. The resulting, aligned traces pave a solid uneven wall (close-up_{wall}); In the third example, a single agent is equipped with two rules, one to establish a continuous trace (rule^0_{tree}) and the second one triggering periodic branching to two sides (rule^1_{wall}). The repeated branching process (process_{tree}) yields a tree-like structure

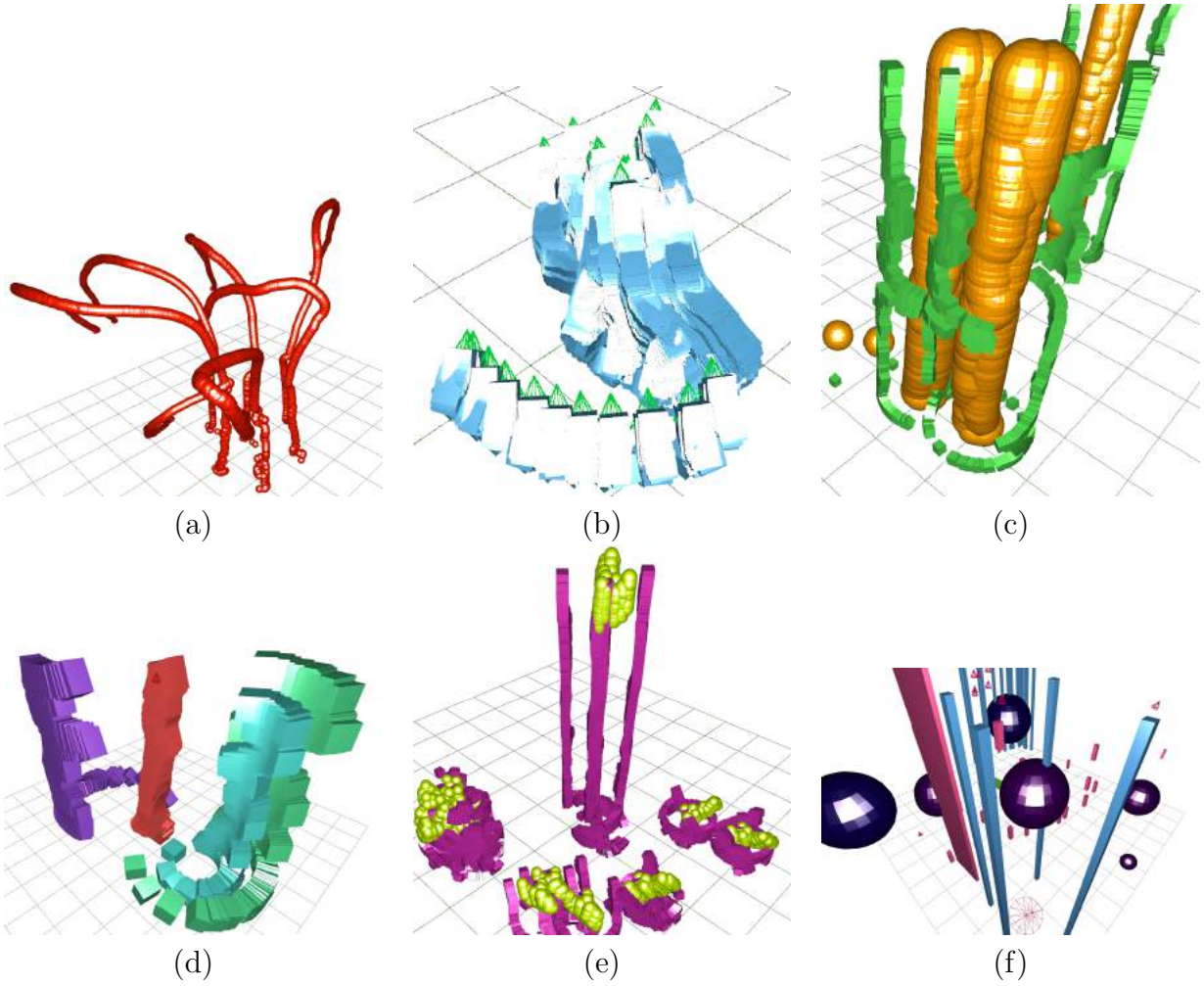


Figure 14.8: (a-b) Flocking SG agents leaving traces of one platonic solid (red spheres and blue cubes, respectively). (c) Individuals with a strong directional upwards urge leave a trail of two solids, golden spheres and offset green cubes. (d-f) More intricate and diverse structures emerge from building efforts by heterogeneous agent sets.

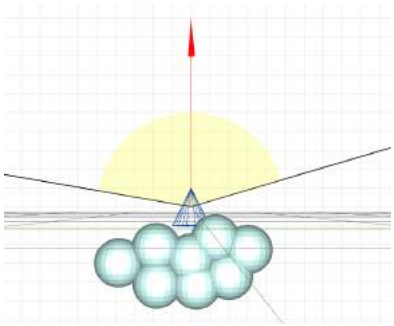
(close-up$_{tree}$).

14.5 Conclusion

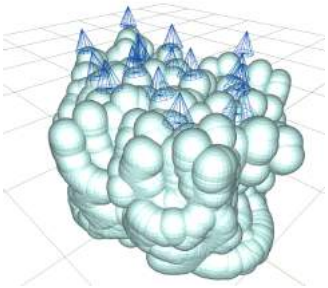
In this paper, we presented an interactive Swarm Grammar simulation. It has been conceptualised and implemented in order to engage a young audience in Artificial Life concepts, namely swarm dynamics and developmental processes. The simulation attempts to intrinsically motivate the users by keeping the learning-curve as low as possible. At the same time, we challenge the users' competence by attaining a relatively expressive programmable representation, including boid flocking behaviour as well as (re-)production behaviour of Swarm Grammars. The gap between expressive representation and simplicity is bridged by means of visual programming interfaces for configuring the simulation space as well as individual agent behaviours.

We exhibited some of the artefacts designed by a number of high-school students at the age of twelve to fifteen. We suggested several possible improvements to the software based on feedback by the students but also based on observations during supervised hands-on sessions with the simulation. We complemented the display of the students' works by three additional Swarm Grammar examples that explicitly rely on (a) multiple construction elements in single rules, (b) differentiated reproduction, and (c) branching production rules.

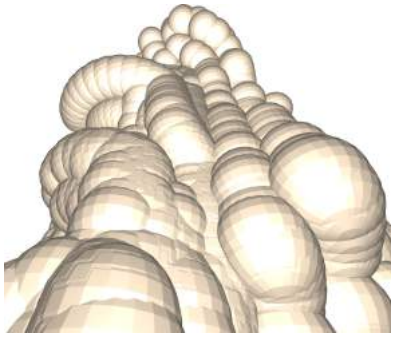
In order to further the presented work, we suggest to translate all boid parameters into meaningful interactive visuals. For instance, the line-width of arrows representing various flocking urges could stand for the according, relative weights. Field of View and other visually represented parameters should be readily manipulatable, and not only be altered by means of textual GUIs. The maximal Age of an agent could be visualised by projecting a faded out geometry along its current trajectory. Also, the composition of (re-)production rules could be improved by relating them to the actual environment: the production objects could be projected on top of the actual simulation environment in order to facilitate precise definitions of environmental alterations. Programmatically, we suggest switching from the current, class-based architecture to a component-based perspective that allows to aggregate behaviours. This would simplify the template management and provide the flexibility to assign behaviours to arbitrary objects, as asked for by the students.



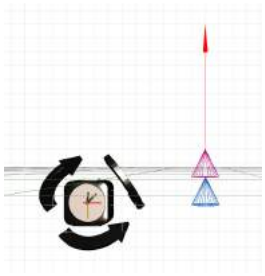
(a) rule$_{cloud}$



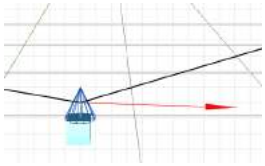
(b) process$_{cloud}$



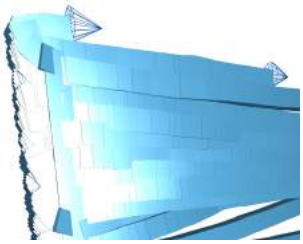
(c) close-up$_{cloud}$



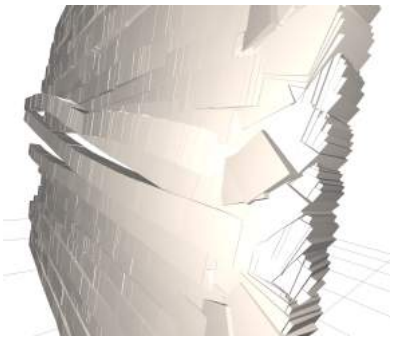
(d) rule$^0_{wall}$



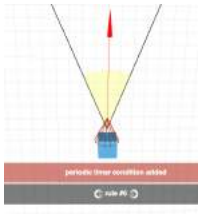
(e) rule$^1_{wall}$



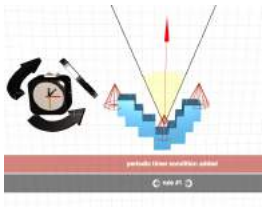
(f) process$_{wall}$



(g) close-up$_{wall}$

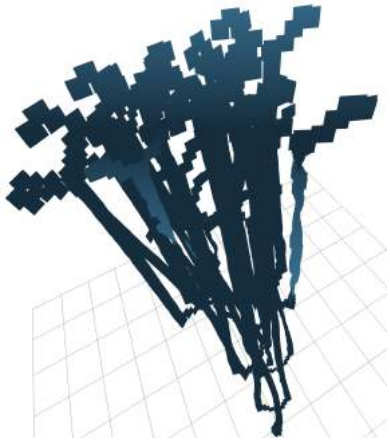


(h) rule$^0_{tree}$



(i) rule$^1_{tree}$

(j) process$_{tree}$



(k) close-up$_{tree}$

Figure 14.9: (a) An unconditional multi-solids production rule to trace an upwards-flocking agent. (b) A snapshot of the resulting developmental process, given a small set of initial agents. (c) A close-up of the final artefact. (d) An agent that flies upwards without considering any distractions and periodically produces (e), an agent heading to the right and leaving a cubic-trail behind. Its trajectory is influenced by its neighbours and chance. (f) The resulting developmental process, and (g) the final artefact close-up. (h) A simple trail production rule, combined with (i) a periodic branching rule, resulting in a (j) branching developmental process. (k) The final artefact set in scene with a pixelated 2D style.

Chapter 15

Component-Based Networking for Simulations in Medical Education

For the purpose of medical education, our research team is creating LINDSAY, a 3-dimensional, interactive computer model of male and female anatomy and physiology. As part of the LINDSAY—Virtual Human project, we have developed a component-based computational framework that allows the utilization of various formal representations, computation engines and visualization technologies within a single simulation context. For our agent-based simulations, the graphics, physics and behaviours of our interacting entities are implemented through a set of component engines. We have developed a light-weight client/server component, which spreads its siblings in the system's component hierarchy over a wireless or wired network infrastructure. In this paper we demonstrate how our client/server component paves new ways for organizing, generating, computing and presenting educational contents.

Sebastian von Mammen, Timothy Davison, Hamidreza Baghi, and Christian Jacob. *Component-based networking for simulations in medical education.* In: *IEEE Symposium on Computers and Communications, ISCC10*, IEEE Press, 2010, pp. 975–979.

15.1 Introduction

As health care systems all over the world struggle to provide affordable and comprehensive care, the need for excellent education and training of medical staff is more important than ever.

At the same time, emerging digital technologies render it possible to present complex contents faster and better to large audiences than ever before. In this context, we are developing LINDSAY—Virtual Human, a project at the intersection of Computer Science, Education, and Medicine. LINDSAY provides a collection of computational tools for research and learning in the context of human anatomy and physiology.

As a starting point for this project, we experienced and analyzed lectures in human anatomy given by distinguished instructors. These investigations led to the conclusion that the mere projection of a virtual three-dimensional human body in combination with the ability to easily navigate and label the display could greatly facilitate and improve the learning experience. Consequently, such an application became the first milestone of our LINDSAY project.

To this end, we devised a component-based framework to make our content presentation platform extensible [292, 291]. Hierarchies of components can combine attributes and behaviours on different levels of scale and resolution. They can host information for various visualization techniques (e.g., charts, animations), for interaction interfaces (associated with hardware devices or software user-interfaces) or for simulations computed in real-time and driven by heterogeneous computational engines (physics engines, differential equation solvers, etc.).

In this work, we present the design and implementation of a networking component embedded into our content delivery system. It enables a range of network topologies and configurations that can support novel means of active learning in classrooms supported by wireless devices [568].

The remainder of this paper is structured as follows. Section 15.2 introduces the emerging technology of virtual human anatomies in learning environments. It also touches upon agent-based and object-oriented simulation techniques as they are our current focus for the presented networking technology. We also provide references to previous networking solutions for component-based architectures. Section 15.3 describes the design and implementation of our client/server networking component and its integration into the larger component framework of our LINDSAY Virtual Human system. Section 15.4 describes examples of distributed computing and visualization in the context of content generation and delivery in an educational environment. The article concludes with a summary and exploration of opportunities for future work in Section 15.5.

15.2 Related Work

For more than two decades, scientists have been exploring ways to enhance medical research and education by way of computer-based renderings of human anatomy and physiology. The American National Library of Medicine started as early as 1989 with the composition of a comprehensive imagery database of human physiology, also referred to as the Visible Human [283]. Since then, these data sets have been inspiring a large number of virtual anatomy projects for research, patient consultation and education [284].

In the context of education, atlases have been composed that promote the exploration of detailed anatomical terminology in a proper visual context [285]. Some systems have been further extended to incorporate data about actual biomedical processes in the human body, for instance genetic processes [286]. Such systems can be modelled and simulated by means of traditional mathematical methodologies or as large sets of self-organizing bio-agents, or *swarm* systems [288, 289].

Relying on a component-based architecture for a content delivery and generation framework provides the freedom to combine any of these computational modelling and simulation approaches [291]. More specifically, a component can be broadly defined as follows [292]:

> **Component** A component's characteristic properties are that it is a unit of independent deployment; a unit of third-party composition; and it has no persistent state.

The design of component-based software architectures has advantages in regard to various application domains. Frameworks for (human) collaborative work can be implemented by brokering groupware components [202]. The coordinated execution of heterogeneous components works for organizing human collaboration as well as complex code bases that comprise large sets of interoperable, reusable software components. For example, a component-based augmented reality framework could manage components for user interfaces, tracking or object modelling [203]. Computer games, too, face the challenge of integrating vast numbers of software components, whether related to contents, providing networking infrastructure or user interfaces [204]. A tutorial for constructing a component-based framework in the context of Massively Multiplayer

Online games is provided in [205]. For practical reasons, multi-facetted game units are defined by aggregating distinct software components rather than by using established methodologies of object-oriented inheritance [207, 5].

An overview of component-based client/server frameworks is provided in [208]. Sets of component-based distributed embedded systems facilitate coordinated interactions [293]. Redeployment of components across a network infrastructure results in improvements regarding service availability [209]. Alternatively, mobile devices can exchange software components in peer-to-peer networks in order to address user requests [210]. The exchange of software components in heterogenous hardware infrastructures might require adaptation of the control over the respective components or of their data. In [294], such an adaptation strategy is presented for transferring components among devices of varying degrees of computational power, e.g., from a desktop/server to a set of mobile devices. In particular, adaptation is realized by specific drivers that serve as middleware to translate the broadcast software components.

15.3 Component-based Simulation & Networking

In our component-based simulation framework, the LINDSAY Composer, a simulation is represented as a hierarchy of components. Figure 15.1 shows a prototypical component hierarchy of a simulation. Darker shaded boxes represent nested components at a lower level of the system hierarchy. Components may be registered with and interface with various computational engines. As an example, Figure 15.2 illustrates how components on a higher hierarchical level, such as the Blood Cell, can aggregate several subcomponents that are registered with their respective component engines. Here, the Blood Cell component has children registered with Graphics and Physics component engines.

In particular, a component engine accesses a component's data, possibly relying on the presence of various sibling components. State updates of the components drive the computational processes. As an example, a Physics component engine updates the Transform component (position and orientation) of the Blood Cell depicted in Figures 15.1 to 15.3 .

Although the aggregation of different components for individual objects results in a flat functional hierarchy [207, 5], a tree hierarchy facilitates object management in large simulation

environments. Parent-child relationships, as emphasized in Figure 15.1, are primarily semantically motivated. They can, however, also play a role in the context of certain modelling representations such as Membrane Computing, where biological systems are described based on inner-compartmental particle interactions and inter-compartmental particle transfers [523]. In such a case, the component hierarchy in combination with the means of object introspection can provide the data structure required for a component engine that works across hierarchies.

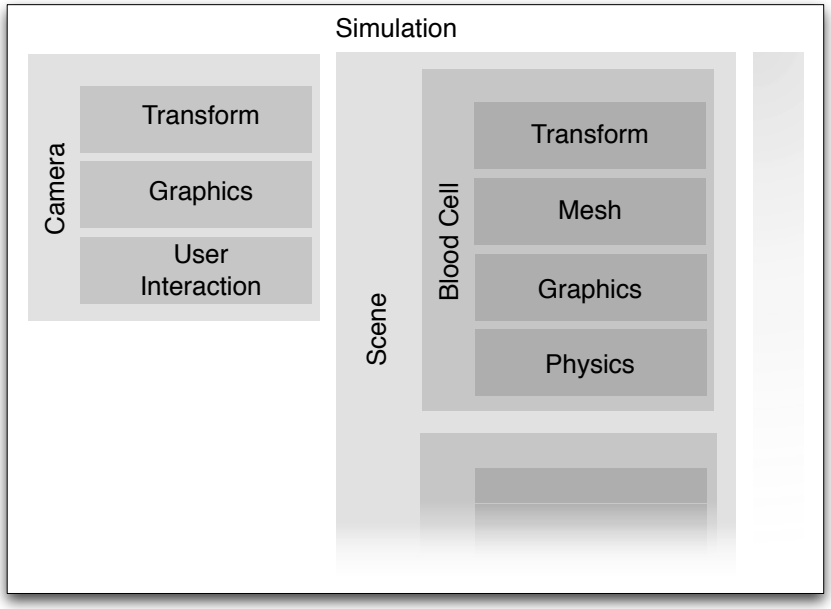


Figure 15.1: Components at different hierarchical levels are defined by the aggregation of their child components. Usually only the leaves of the tree are registered with respective component engines.

15.3.1 Client/Server Components

We have embedded our Server and Client components into the simulation's component hierarchy. Thereby, we determine which parts of the simulation should be broadcast over the network, or where the received components should be embedded in the recipient's simulation context. Figures 15.2 and 15.3 show a Server and Client component within the hierarchies of two prototypic simulation contexts—on a server and a client system, respectively. In accordance with the shaded hierarchy schema introduced in Figure 15.1, the Server component in Figure 15.2 is parented by a Scene component and is a sibling of a Blood Vessel and a Blood Cell.

On the implementation side, our Client/Server component architecture works as follows. The

Server tries to connect with a given Bonjour service name. Once a connection between the Server and the Client component has been established, the Server component first traverses and then broadcasts its sibling components. Next, changes to the properties and hierarchical organization of these siblings are aggregated over each frame of the simulation, and are then packaged and distributed across the network link. On the other side of this link, the Client component first creates, then maintains and updates a mirrored set of these components as it receives updates from the server. These updates also consider structural changes to the hierarchy, including creating, destroying, and rearranging the mirrored components. In addition to the selection of subtrees, a Server component can apply predicates to filter its siblings and their children which results in an arbitrary selection of transmitted and updated components. As an example, one might only be interested in sending component types that encapsulate spatial and visual information but do not process the simulation. Another predicate might filter out a set of components based upon their spatial position within a simulation.

The Client component, on the other hand, skips over the recursive integration of those received components that would require a component engine that is not provided by the client-side system. In the example shown in Figure 15.3, a Client component receives and maintains the Blood Vessel and the Blood Cell components sent by the Server depicted in Figure 15.2. As the Server component has filtered out all Physics components, the client system only processes the visualization of the transferred components, relying on the Graphics component engine and the Transform and Mesh component types.

15.3.2 Technical Aspects

The implemented Client and Server components rely on the Apple Bonjour protocol [569]. A service name is the only information necessary to establish a handshake and a connection link. Mac OS X Snow Leopard on Mac Pro desktop machines with 2.26GHz eight-core Intel processors computed and streamed the physics simulations. Multiple iPhone 3G phones (16GB and 32GB) and iPod touches (3rd generation, 32 GB) were used as wireless devices. In our experiments, we measured about 800KB/s data throughput, rendering the presented configurations more feasible to be deployed on the IEEE wireless standards 802.1g and n, as opposed to b. The Server component's filtering method is implemented based on the NSPredicate class of the Apple Foundation framework [570]. Changes to components are observed via

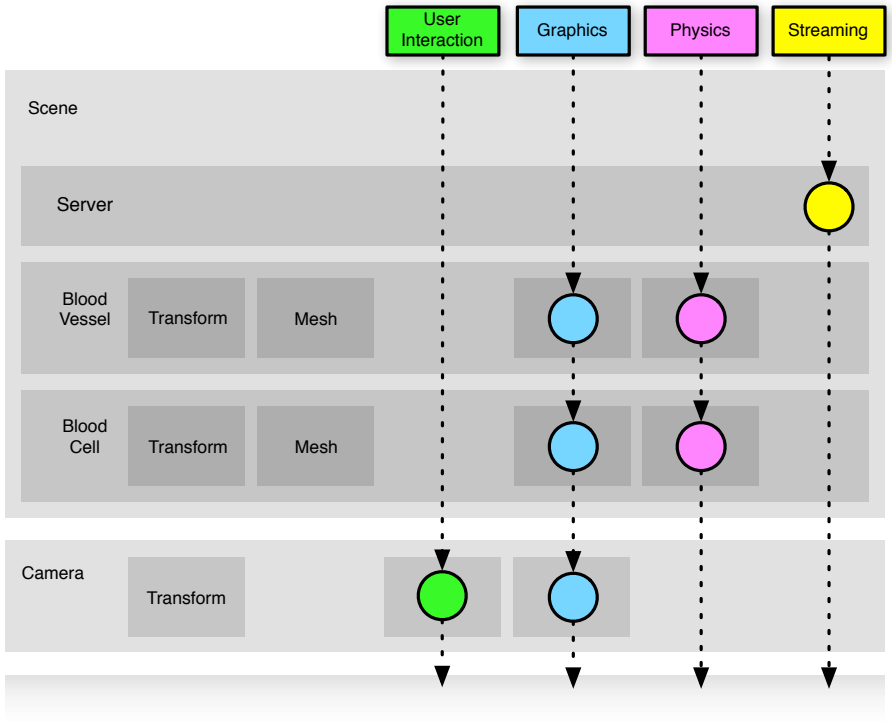


Figure 15.2: A Server component is embedded within the scene of the simulation. It transmits and updates its siblings, Blood Vessel and Blood Cell, across the network. Grey boxes represent the component hierarchies. Circles denote registered components. Component engines (coloured boxes) consider the registered components in the order indicated by the dashed arrows. Parts of the diagram were adapted from [5].

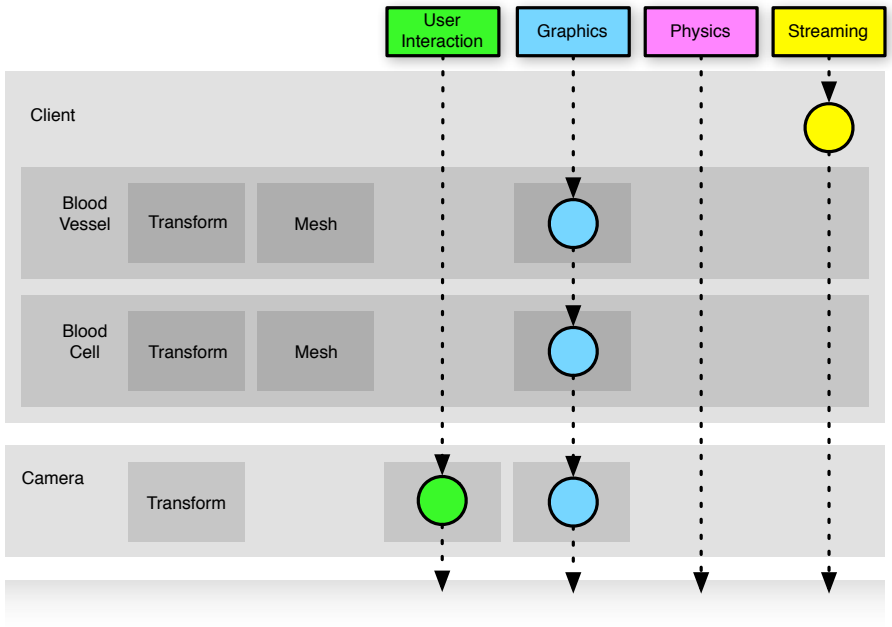


Figure 15.3: A Client component integrates the components it receives as children. In the given case, the Blood Vessel and Blood Cell are stripped off their Physics component as it was filtered out by the streaming server in Fig. 15.2.

the NSKeyValueObserving protocol.

15.4 Example Scenario

The introduced Client and Server components allow for a broad variety of network configurations. In this section, we present an example scenario for the classroom that utilizes these components in different ways. Server and Client components can exist on any device in a heterogeneous network. Multiple Client and Server components may even exist within the same component hierarchy on a single device. This raises several interesting possibilities. For example, one can have a centralized simulation running on a single powerful machine, with less powerful devices connecting to this device to only visualize (not compute) smaller portions of the simulation. Distributed simulation is also possible, with multiple computers computing different pieces of the simulation. In another example multiple handheld devices are connected to a simulation that is run on a powerful shared device, where each handheld controls different portions of that simulation.

We developed an example scenario around a physics-based blood clotting simulation. Figure 15.4 shows a screenshot of the simulation run by the LINDSAY Composer on a desktop computer. One can see a clot forming over the breach in the vessel wall.

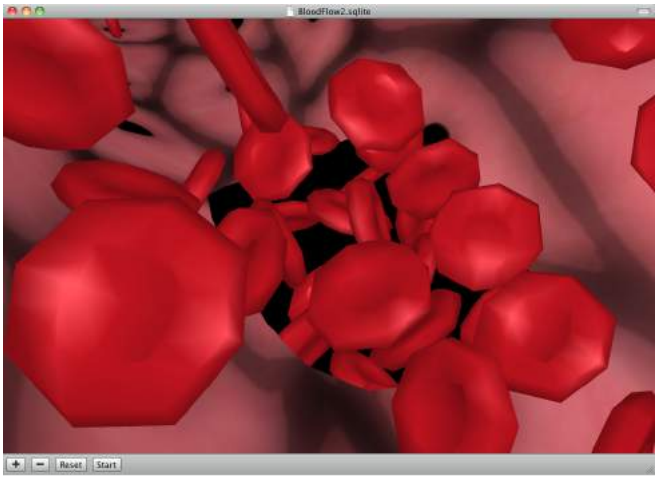


Figure 15.4: Screenshot of a blood clotting simulation run with the LINDSAY Composer. Red blood cells are streaming through a blood vessel.

15.4.1 Distributed Computation

In order to increase the scale of the simulation, the introduced network components can be used to establish a distributed computing network. Figure 15.5 presents a network configuration in which different parts of a simulation are computed on different machines. In addition, one machine is dedicated to retrieve and aggregate the information of the separately computed parts, which are then visualized in a single simulation context. Figure 15.5(a) shows a conceptual diagram of the network configuration: Three Client components load their data into the same simulation context for visualization. The screenshots in Figures 15.5(b-d) illustrate those three independent simulations running on separate machines. Figure 15.5(e) shows the resulting combined visualization in one simulation context.

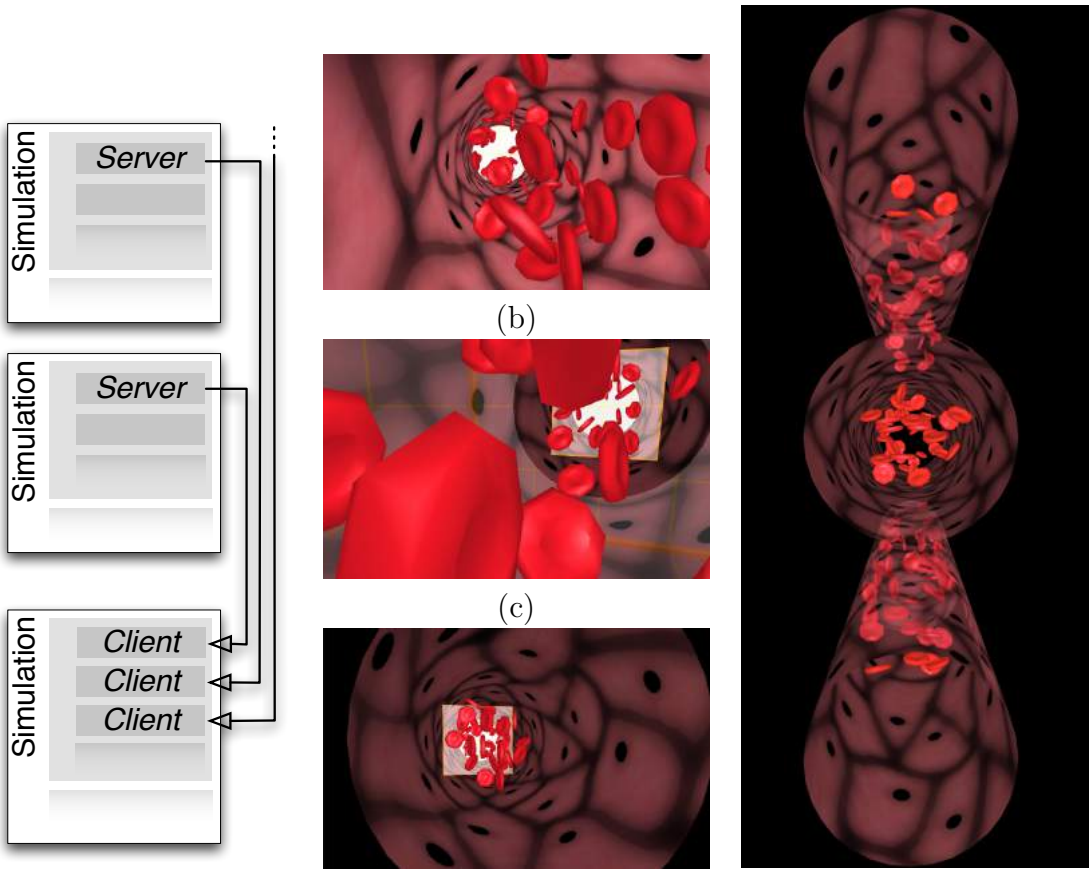


Figure 15.5: (a) One computer receives data sent from three other machines and integrates it into one simulation context. (b-d) Snapshots of the simulation processes on the server machines. (e) The visualization of the merged simulation data on the client machine.

15.4.2 Distributed Learning

Currently, a room for demonstration and development purposes is setup for the LINDSAY project that is similar in size to a medium-sized classroom (ca. 20 students). Simulations of human anatomy are projected onto a large backlit projection screen. This setup is designed so that one lecturer can teach the students while guiding them through various simulation contents. The Server/Client component tandem can, however, add a new dimension of student involvement to the classroom experience. In particular, students can connect with their wireless handhelds, cell phones, laptops or tablets and explore the shared simulation space by themselves. This means that each student could take a different perspective of following the shared simulation. For our example scenario, students could observe the blood clotting from inside or outside the blood vessel, hop onto a blood cell and follow the action, while the rest of the class simply join the instructor's perspective. Alternatively, wireless devices can be used as remote controls for steering the actual scene presented on the screen—whether an inquiring student or a guiding teacher controls the simulation remotely would depend on the educational context and the stage of the class.

Figure 15.6(a) illustrates the network configuration for such a classroom setup. The projected simulation (top box) streams simulation data to a number of wireless clients. Among them is an iPod that is used as a remote control (middle box) whose directions are fed back into the simulation. As complex physics simulations are typically not suited for handheld devices, we only send the visualization data of the simulation to the iPhone/iPod clients (Figure 15.6(b)). In Figure 15.6(c), an on-screen joystick is drawn on top of the visualization of the simulation. It can be used to control the camera on the iPhone. In particular, the virtual joystick can be used to move the camera forward, backward, left and right. Additionally, the camera can be rotated by touching on the screen and dragging the fingers in the desired direction—up/down pitches the camera, whereas left/right rotates the camera around its y-axis.

Since the camera and its transformation is transferred to the workstation, any change to the camera on the iPhone will change the camera on the workstation. Hence, the camera of the main simulation running on the workstation can be controlled using an iPhone, which would typically be used by the course instructor.

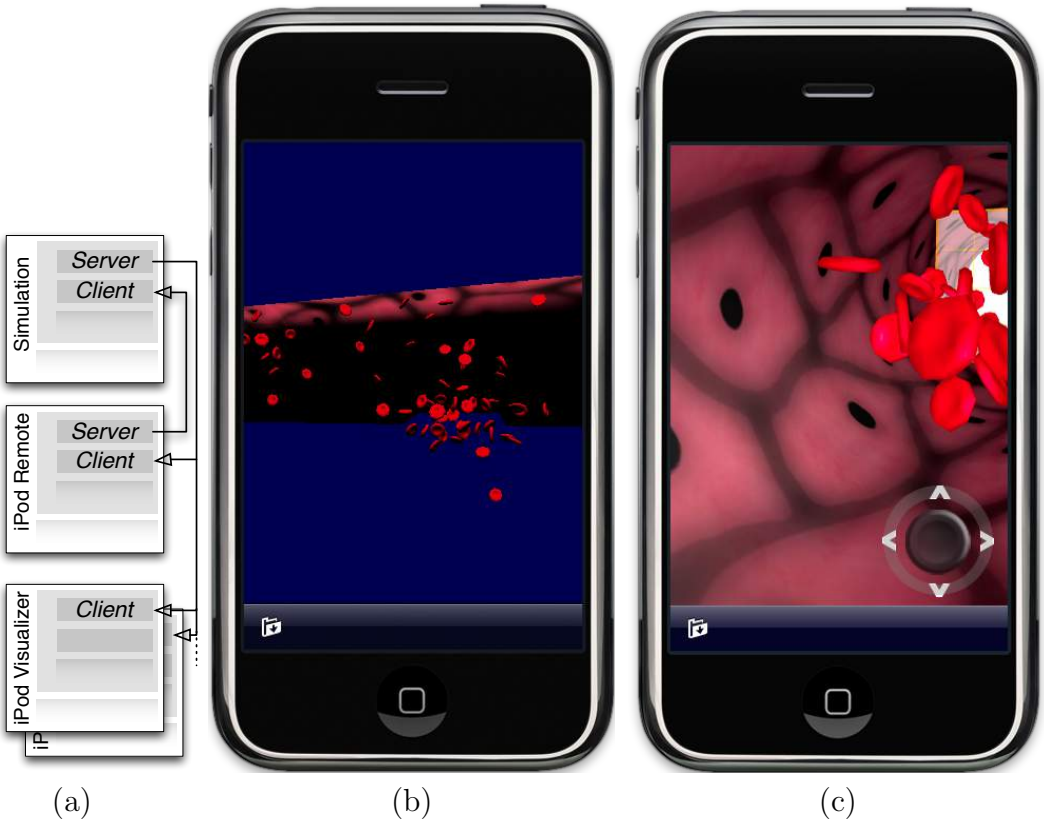


Figure 15.6: (a) A network configuration for an interactive classroom: A shared simulation is presented on individual wireless devices. Information from one of these devices is fed back into the simulation. (b) Visualization of the simulation on an iPhone—as described in Section 15.3.1, Physics components are not transferred. (c) On a second iPhone, a virtual joystick (in the bottom right corner) is used to guide the camera.

15.5 Summary & Future Work

We introduced the LINDSAY Composer as a generic, component-based simulation platform. Its primary purpose is the simulation and visualization of complex biomedical systems for teaching in medical education. We present Client and Server components that can be embedded in the hierarchical data structure of simulation within the LINDSAY Composer. These networking components can be easily used to send and receive parts of a simulation over a network. Since multiple Server and Client components can be integrated into one system, and since they are able to handle arbitrary component data, complex networking scenarios are possible.

With several examples, we have demonstrated the flexibility and simplicity of using Server/Client components to implement interesting networking scenarios. These scenarios are designed for, but not limited to a classroom context. In particular, we showed how a complex physics-based simulation can be computed on several machines and how the results can be visualized by one client. Filtering out Physics components from the transmitted simulation data renders it possible to have complex simulations visualized on wireless devices with limited computational power. In particular, we stream visualization data from a blood clotting simulation to a set of iPhones/iPods that are used by students to individually explore a shared simulation. By adding a Server component to a wireless device, it can be used as a remote control and feed data back into another system. In an example, we used an iPhone to direct the camera of a remote simulation.

The combination of a component-based simulation framework and easy-to-use networking components allows for a vast number of possible networking scenarios. In addition to the presented examples, students could use networked devices to simultaneously manipulate the data of a shared simulation. In this context, the idea of turning iPhones into augmented reality devices might prove invaluable: depending on the environment captured by an iPhone's camera, additional information displayed on the handheld devices could overlay the projected simulation. Mixed-reality elements could be introduced, for instance, using the overlay technique to display simulation data in the context of a human body.

In addition to exploring new technological prototypes for interactive classroom settings, an easy-to-use graphical user interface will have to be provided so that lecturers can quickly design and

setup distributed simulation scenarios. The Client/Server components should be extended by an authentication mechanism. This could help to maintain confidentiality about certain course material, which is required, for instance, in the context of human corpses. Authentication would also empower the instructor to individualize the learning experience of the students, e.g., by assigning different means to manipulate simulation data, or by giving access to different supplementary data sets. Associating specialized configurations with handheld devices could also support group work.

Chapter 16

EvoShelf: A System for Managing and Exploring Evolutionary Data

Systems that utilize evolutionary computation produce large amounts of data. Quite often, this data has a convenient visual representation. However, managing and visualizing evolutionary data can be a difficult and onerous task. By employing techniques used in photo management software, we have produced a system that helps to visualize and organize evolutionary data while retaining a complete record of a simulation. By means of a simple plugin architecture this system can be extended to import data produced by arbitrary evolutionary systems. We present the system's architecture, its features, and we provide a comprehensive example, highlighting its advantages in applied research.

Timothy Davison, **Sebastian von Mammen**, and Christian Jacob. *Evoshelf : A system for managing and exploring evolutionary data*. In: *Proceedings of Parallel Problem Solving in Nature (PPSN)*. Springer Verlag, 2010, pp. 310–319.

16.1 Introduction

Evolutionary systems produce large amounts of data. Beyond the obvious data (such as the genotype and phenotype of an individual), there is a considerable amount of meta-data produced as well. Such data includes the hereditary data, fitness values, and other attributes of the evolutionary computation approach being employed.

It is common to manage experimental data by means of a file-system browser, such as the Finder in Mac OS X, and Windows Explorer in Microsoft Windows. Searching or organizing individuals according to various criteria is a laborious task in such systems. Consider a system that organizes the individuals produced by an experiment into sub-directories by generation, giving each individual its own file containing its genotype, and phenotype, along with meta-data such as fitness, or genealogy. Filtering these individuals by fitness value would be a difficult task with either file-system browser.

An evolutionary system may employ an interface of its own for browsing the data that it produces. In this case, the visualization procedures and the management of the genotype/phenotype data are typically implemented specifically for the one evolutionary system. However, the universality of evolutionary algorithmic approaches renders generic visualization and data management techniques valuable across various application domains.

In a way, the situation is very similar to managing individual (digitized) image and music collections. Such libraries can easily consist of thousands of items. A number of applications have made the organization and management of such data much easier for the end user [571, 572, 573, 574]. We propose a system that can deal with evolutionary data with the same ease of use and flexibility as provided by these mainstream media management applications.

EvoShelf is an extensible system that allows for the importing and exploration of evolutionary data from evolutionary systems. It solves the challenge of organizing imported metadata by providing a navigable, and searchable image-based browser that uses interface design elements from Apple's photo and music management software iPhoto [571], and iTunes [572]. Furthermore, it provides a plugin framework for building additional import modules and visualizations.

In Section 16.2 we explore the topic of visualizing data in evolutionary systems. Section 16.3 presents the design of the *EvoShelf* system and its graphical user interface. It also touches upon some of the visualization techniques included in *EvoShelf*, as well as details about its plugin architecture. In Section 16.4 we use *EvoShelf* in coordination with an existing evolutionary system for semi-interactive evolutionary computing, and for analyzing the results produced by that system. We will conclude in Section 16.5 with a summary of our work, along with possible directions in which to take it in the future.

16.2 Related Work

We briefly outline the data management and user-interface approach of various software that inspired the *EvoShelf* visualization and management system. Secondly, we outline various techniques that have been developed for visually supporting computational evolutionary experiments.

16.2.1 Digital Media Libraries

The framework presented in this article was mainly inspired by iPhoto, Apple's mainstream photo management application [571]. It is capable of organizing and browsing thousands of photos. Despite the large amounts of information that it is capable of presenting to the user, it maintains a very simple and intuitive interface. It consists of two primary views, an organizer view, and an image browsing view (Figure 16.1(a)). Multiple images, up to and including an entire library of photos, are displayed in the browsing view. The organizer view is used to filter this view into subsets of photos, such as those represented by a photo album containing the user's favorite photos. As photos are imported into the system they are grouped into events. Pictures taken during a certain period of time might have all been taken during a vacation and the respective group of photos could be labeled after the location of the recreational stay.

iTunes is another application from Apple Inc. that manages a large amount of data in a similar fashion to iPhoto. Unlike iPhoto, whose interface is focused on visualizing and managing photos, iTunes is targeted towards playing music and organizing large digital music collections. Visual cover art often decorates individual music files, but the iTunes library is mainly organized by sorting and searching through textual meta-data such as artist name, music category, or album name (Figure 16.1(b)). Together, iPhoto and iTunes suggest an interface that combines visualization and meta-data management techniques that could be very powerful for organizing evolutionary data.

(a) (b)

Figure 16.1: The user interfaces of the media management applications (a) iPhoto and (b) iTunes.

16.2.2 Evolutionary Visualization Techniques

Various data visualization techniques have been presented in the context of evolutionary computing. On the one hand, individuals can be compared at a glance based on their multi-dimensional genotypes, independent of the respective interpretation or phenotype. On the other hand, methods of visualization have been developed that capture characteristics of whole populations, allowing one to visually track the evolutionary process.

Pohlheim, for instance, presented a toolkit of convergence diagrams, 3D line plots, and 2D image plots, to visualize the evolution of fitness values and other individual attributes. Hart and Ross introduced a tree-based visualization to trace the ancestry of the best individual produced by an evolutionary run [575]. Daida et al. unfold genetic ancestry onto concentric circles on a 2D plane to create a compact and highly scalable visualization [576]. Wu et al. represent genotypes as sequences of color coded stripes whose colors correspond to different genes [577]. Keim et al. designed a system to visualize search queries on a (relational) database [578]. Data items that match the query most closely are arranged in the center of a spiral arrangement. This visualization technique can be used to relate individuals in an evolutionary system in arbitrary ways, e.g. by comparing fitnesses or individual attributes. In [579], Khemka and Jacob have closely investigated the possibilities to visualize population-based optimization processes at various levels of scale—from the individual to sets of experiments. They provide an easily adaptable user interface with various interactive manipulators to explore optimization processes

across these scales.

16.3 The *EvoShelf* System

The interface of *EvoShelf* is divided into three window panes (Figure 16.2). The organization view on the left-hand side is used for selecting and grouping imported experimentation data (Section 16.3.1). The user's selection is shown in the browser view in the center pane. An inspector view (right-hand side) shows further details about an individual or an experiment. In addition to importing and inspecting functions, the toolbar at the top of the window gives access to built-in visualization methods which are explained in Section 16.3.2. Typically, a user of *EvoShelf* writes a plugin to import and visualize data for his respective evolutionary system, if it does not already exist. We provide details about plugins in Section 16.3.3.

EvoShelf makes use of lazy fetching of data. That is, images and attributes of an individual are not loaded until they are needed (such as when the user scrolls to them). When individuals go off screen, their data is unloaded. In this way, we have manipulated data sets with over $40,000$ individuals. Conceivably, *EvoShelf* can work with even larger datasets. To further increase the scalability of *EvoShelf*, high resolution images of individuals are loaded on demand—if no zoom is required, a low resolution image is displayed instead.

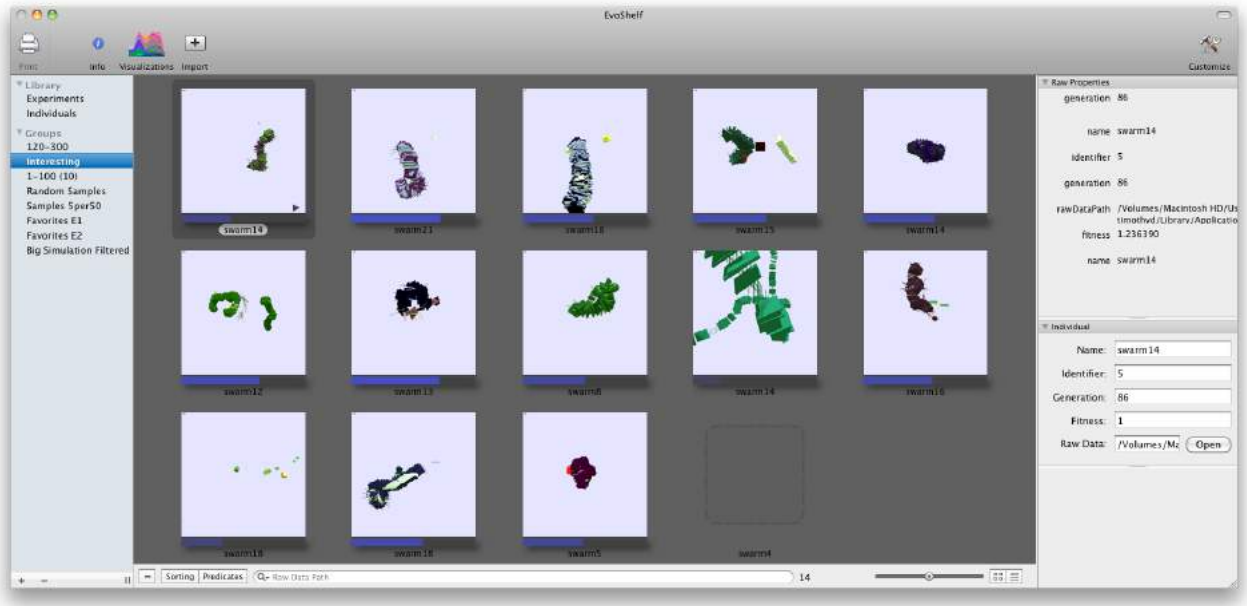


Figure 16.2: The graphical user interface of *EvoShelf*.

16.3.1 Individuals, Experiments, and Groups

The organizational view to the left of Figure 16.2 is divided into a library section and a groups section. In the library section, the user can select either *Individuals* or *Experiments*. In particular, the *Individuals* selection displays the images of all the individuals in the library, whereas *Experiments* shows representative thumbnails of all the imported experiments. The user can browse through the set of individuals of any experiment by hovering with the mouse over its thumbnail. The individuals of an experiment are revealed when the user double clicks on the experiment.

The data in the browser view can be sorted or filtered by the experiments' and individuals' attributes. Once the user has formed a suitable selection he can save his selection in a group, which would be equivalent to photo albums or playlists (as in[571, 572]). In Figure 16.2, a group labelled *Interesting* is selected, which hosts individuals from multiple experiments that the authors found of interest. Groups can be organized hierarchically. That is, one can form groups containing groups. When such a group is selected a union is formed from all of the individuals contained within the subgroups.

The controls at the bottom of the interface allow the user to remove individuals from a group or from the library, to sort individuals, to search for individuals according to arbitrary attributes (such as fitness value or generation), to scale the size of the images displayed, and to change the display mode. One display mode shows individuals as a collection of images, another one lists them in tabular format. The latter view is convenient for sorting and searching through individuals based upon numeric or textual attributes.

One individual is selected in the browser view in Figure 16.2. The image representing the individual was generated by the evolutionary system used as a test run for *EvoShelf* (see Section 16.4). In the given case, a play button (a right pointing arrow) allows one to re-run the simulation that produced and/or evaluated the selected individual. The button is not shown if the plugin for the particular evolutionary system does not support this option, and it only appears when the user hovers the mouse over the image.

Below the images in the browser view in Figure 16.2, blue bars represent the individuals' fitnesses. The bar is scaled to the minimum and maximum fitness of all the individuals currently

displayed in the browser view. The higher the fitness, the brighter and longer the bar. No image is provided for *Swarm35* indicating that the genotype data was successfully imported but no image was found—in the given case, the simulation was terminated before a screenshot would have been taken.

The inspector view displays several default properties about the imported data, such as the file name of an individual or experiment. A custom interface for the inspector can be defined via the plugin architecture (Section 16.3.3).

16.3.2 Built-in Visualization Techniques

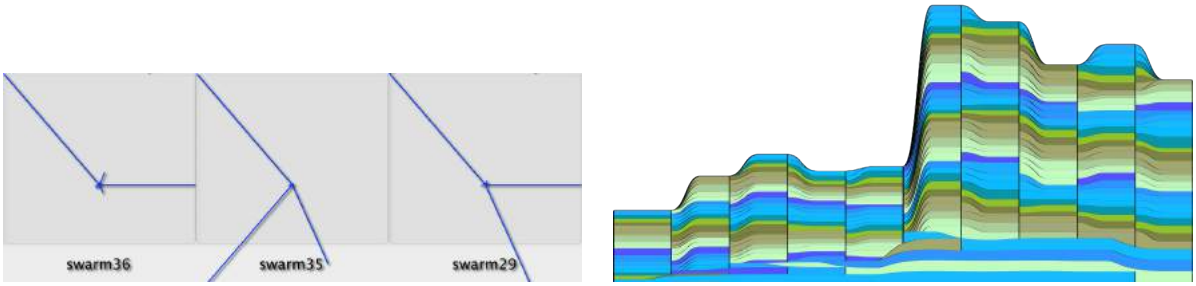
EvoShelf employs two basic built-in visualization techniques: star plots of name-value pairs [579] and FitnessRiver, a derivative of the ThemeRiver^{TM} method, which integrates local numeric values with global trends [6].

A star plot in *EvoShelf* visualizes a set of name-value pairs as a series of radially arranged line segments (Figure 16.3(a)). The length of a line segment is representative of an attribute's value and it is normalized to the attribute's maximum value in respect to the selected individuals. An attribute's line segment will consistently appear at the same location in a star plot to render individuals comparable.

The ThemeRiver^{TM} visualization method produces a stream diagram that is read from left to right. Currents in the stream represent individual themes that occur, grow and decay over time. Instead of separating equivalent attributes into individual currents of a stream diagram, our FitnessRiver visualization method stacks the fitness values of individuals on top of each other. The fitness of an individual is proportional to the width of its current. Different colors are used to distinguish between successive individuals. Discontinuing currents indicate the removal of an individual from the evolutionary process. In the FitnessRiver visualization the x-axis represents the sequence of generations. A flat baseline is used so that the user has a greater sense of the progression of the fitness evolution (Figure 16.3(b)).

In Figure 16.3(b) we can see a large jump in the overall fitness at about the middle generation. When we look closely, we see that there are a few very successful individuals in the previous generation. We can see how these individuals likely contributed to the next generation. Fur-

thermore, the majority of individuals in the new generation have noticeably more fitness than those in the previous generation.



(a) (b)

Figure 16.3: (a) Individuals are comparable based on their star plots. (b) The Fitness-River visualization shows the evolution of local and the global fitness. It is an adaptation of ThemeRiver^{TM} [6].

16.3.3 Plugins

A user can define additional import modules, visualization modules, data models, and finally custom inspector views for custom data models[1]. A few basic classes are provided for these modules and models that serve as plugin templates. The importing process, including control over import dialogue windows, can be adapted and alternative visualization modules can be subclassed from the*EvoShelf* visualization view controller class.

The default data model (Figure 16.4) is well suited for evolutionary algorithms (EA) and other forms of heuristic computation, such as particle swarm optimization (PSO). For instance, each step in a PSO simulation could correspond to a generation in an EA. This could however, generate a significant amount of individuals, in which case one might prefer to only import the final individuals from the system. In order to adapt the data model for differently organized

[1] *EvoShelf* plugins are written in Objective-C and should use the Cocoa API [580].

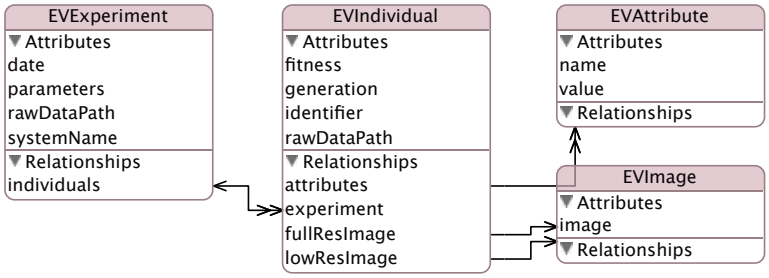


Figure 16.4: The default data model for importing and managing *EvoShelf* data.

information, the *EvoShelf* data model needs to be adapted. For instance, the attributes for the classes *EVExperiment* and *EVIndividual* need to be adjusted to fit the given experiment. The new attributes automatically determine the searching and sorting options in *EvoShelf*, as well as the information provided by the inspector view. In case a more elaborate inspector view is desired, an interface constructed in Apple's WYSIWG Interface Builder application can be loaded.

16.4 Example Scenario

In this section, we explore the use of *EvoShelf* with a preexisting evolutionary system. In the evolutionary system of choice, Swarm Grammars (SGs) are bred by means of a Genetic Programming algorithm to produce architectural idea models [345]. SGs are a swarm-based developmental model in which production and interaction rules guide the movements, constructions and the reproduction of agents in 3D space.

In a subdirectory for each generation, the genotypes are stored as text files and snapshots of the corresponding phenotypes as images. Fitness evaluations for the individuals are stored in an additional file. When importing all the individuals, including their image representations and their meta-data into *EvoShelf*, the original directory structure is automatically copied into *EvoShelf*'s database.

Figure 16.5(a) shows a set of interesting SG specimens. We want to emphasize that due to their partially very low fitness values (swarms 3, 6, 7, and 18), we would have very likely not inspected these phenotypes without relying on *EvoShelf*'s visual browsing functionality. Based on these undervalued, interesting phenotypes, we were able to improve the fitness function that drives the SG evolution. In particular, we shifted the geometrical focus of the fitness evaluation in respect to the SGs' constructions to better suit the favored ones.

We also used *EvoShelf* for a semi-interactive evolutionary process by repeatedly selecting and exporting interesting individuals, modifying the fitness function and parameters to the GA, breeding their offspring for a fixed number of generations and importing the outcome (Figure 16.5).

We discovered that the SG GP evolution usually converged prematurely after at most several

(a) (b)

Figure 16.5: (a) 20 interesting individuals are selected from an experiment and served as the initial generation for a (b) follow-up experiment.

hundred iterations. Figure 16.6 shows the FitnessRiver plot over 300 generations. Overall 20,000 individuals were computed and imported into *EvoShelf*. We noticed that the overall fitness of our individuals had stagnated by the $100th$ generation (there is a very slight improvement in fitness past this point). Figure 16.7 confirmed our assumption of over-fitting: Up to the fitness stagnation at around generation 100, we randomly chose and plotted one of the ten best individuals every ten generations. For the period afterwards, we plotted one of the ten best individuals at random every 20 generations. And indeed, the phenotype images in combination with the star plots reveal a one-sided development, most easily recognizable by the inverted T-shaped star plots. Upon closer investigation, this similarity corresponds to the deployed amounts of two out of three construction elements provided to the SG agents (rods and layers), and the amount of construction elements that were placed outside of the intended target area. As the latter construction elements reduce the fitness of an individual, their increase might explain the fitness fluctuation as observed in Figure 16.6.

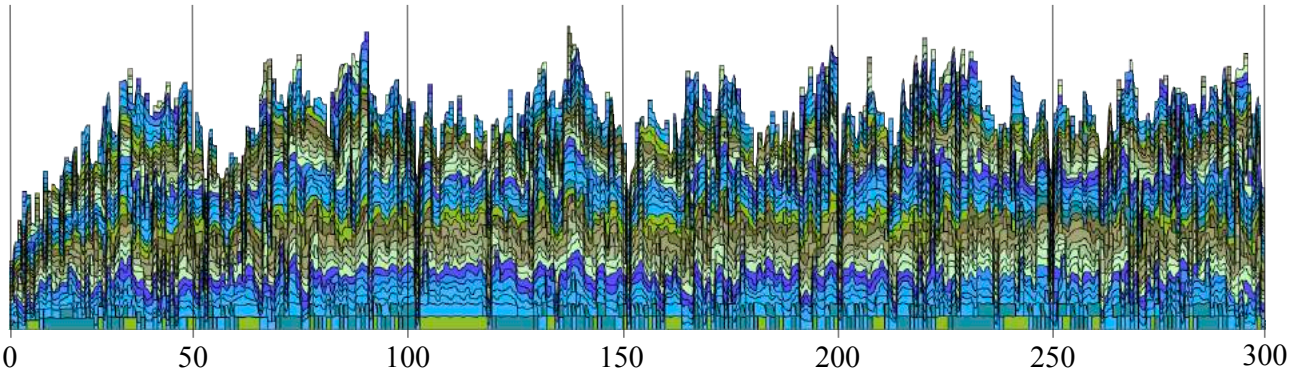


Figure 16.6: The FitnessRiver plot shows stagnating and fluctuating fitness development after about 100 generations. The vertical lines denotes each $50th$ generation.

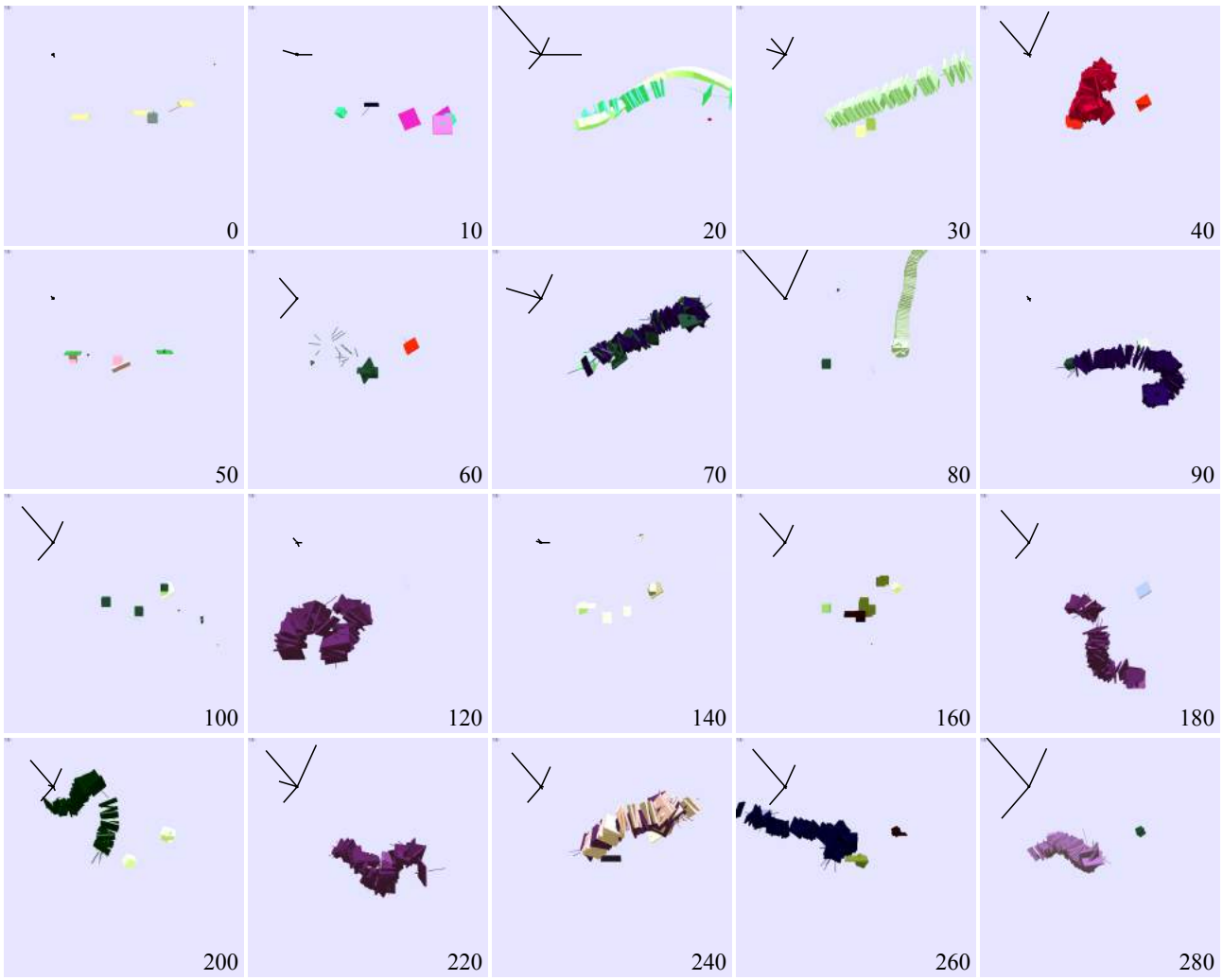


Figure 16.7: First, every ten generations, then (2*nd* half) every 20 generations, a star plot and phenotype of a randomly selected individual is shown.

16.5 Summary and Future Work

We presented *EvoShelf*, an easy-to-use application for managing experimental data produced by arbitrary evolutionary systems. *EvoShelf*'s user interface is similar in its simplicity to mainstream media-browsers. Fast browsing of supplementary images associated with each specimen or of generic visualizations of the individuals, for instance by means of star plots, enables the user to retrace and interactively explore vast amounts of data produced evolutionary experiments. Storing, retrieving and ordering experimental data is facilitated by a simple yet powerful search function that considers the specimens' attributes and meta-data (generation, fitness, etc.). Hierarchical grouping structures further facilitate the management of large amounts of experimental data. In addition to the built-in management and visualization meth-

ods, *EvoShelf* can be extended by plugins that implement the required import, visualization or introspection functionalities. According programming templates are provided that can be easily adjusted or majorly extended, depending on the user's demands.

We applied *EvoShelf* on an evolutionary application that breeds Swarm Grammars to generate architectural idea models [345]. Due to the convenient and fast browsing functionality of *EvoShelf*, we have been able to identify specimens that received low fitness values despite their appeal. As a consequence, *EvoShelf* helped us to adjust the fitness function of the SG GP system to better suit our expectations. By means of the visualization techniques that come with *EvoShelf*, FitnessRiver and star plots, we have been able to track and investigate an over-fitting process in our evolutionary runs. Finally, by using the selection and storing capabilities of *EvoShelf*, we have been able to introduce interactivity into an otherwise autonomous evolutionary process.

In the future, we would like to add more visualization plugins, as well as extend the current visualizations. Several improvements are possible in respect to the deployed visualization techniques. For instance, it should be possible to overlay different individual-based visualizations as we have done in Figure 16.7. The star plot visualization that we applied should be extended to improve its readability— possibly by an underlying, grayed out, partitioned circle, different coloring schemes, or line strengths. Overall, we found it would be useful to automatically associate representative specimens with global trends, as attempted by the combination of Figures 16.6 and 16.7. The FitnessRiver visualization could possibly be extended to also track the application of genetic operators and the course of inheritance. We would also like to explore importing data from an evolutionary system as it runs. Taking this a step further, one could also use *EvoShelf* as the basic user interface for controlling a system that uses interactive evolution as found in [579].

Part IV

Optimising Models of Interactive Self-Organising System

Chapter 17

Optimization of Swarm-based Simulation

In computational swarms, large numbers of reactive agents are simulated. The swarm individuals may coordinate their movements in a "search space" to create efficient routes, to occupy niches or to find the highest peaks. From a more general perspective though, swarms are a means of representation and computation to bridge the gap between local, individual interactions and global, emergent phenomena. Computational swarms bear great advantages over other numeric methods, for instance regarding their extensibility, potential for real-time interaction, dynamic interaction topologies, close translation between natural science theory and the computational model, and the integration of multi-scale and multi-physics aspects. However, the more comprehensive a swarm-based model becomes, the more demanding is its configuration and the more costly its computation. In this article, we present an approach to effectively configure and efficiently compute swarm-based simulations by means of heuristic, population-based optimization techniques. We emphasize the commonalities of several of our recent studies that shed light on top-down model optimization and bottom-up abstraction techniques, culminating in a the postulation of a general concept of self-organized optimization in swarm-based simulations.

Sebastian von Mammen, Abbas Sarraf Shirazi, Vladimir Sarpe, and Christian Jacob. *Optimization of swarm-based simulations.* ISRN Artificial Intelligence Article ID 365791 (2012), pp. 1–12.

17.1 Introduction

Agent-based modelling techniques have prepared the stage for the systematic exploration of complex systems. The interconnection of multiple simple, state-based units, as propagated in cellular automata [340] or random boolean networks [581], yields complex, a priori unpredictable but iteratively computable system behaviours. Discretization and confined interaction spaces have rendered a systematic and comprehensive investigation possible that has provided far-reaching conceptual insights—most prominently the identification of complexity classes and the provision of tools for the classification and analysis of complex systems [341, 299].

Taking the alternative route and trying to consider and integrate even minute details unearthed by natural scientists and amalgamating them into one comprehensive computational model is a daunting task. Yet, steps in this direction have been successfully taken. Material scientists have paved the road in the field of multi-scale model integration in order to gain insights into the properties and behaviours of compound materials [269]. Biomedical researchers have recently been taking similar approaches that target numerous scales of human physiology—from the level of gene expression up to a human population [87]. The integration of model data different from traditional equation-based systems is also moving forward. Recent trends in developmental simulations, for instance, integrate high-level agent behaviours, such as morphogenesis or proliferation, and physical environmental constraints [582, 583, 584, 585]. Although these simulations are typically confined to lattice spaces, often even to two spatial dimensions only, they show considerable promise in retracing natural phenomena of growth and physiological development.

Unfortunately, one inevitably faces a trade-off between real world phenomena and the intricacies of the corresponding models, between the number of interdependent variables and computational viability—in terms of computational efficiency and of effectiveness regarding the expected results. Agent-based models scale particularly poorly with increasing degrees of interaction and increasing numbers of simulated agents. Due to their numerous advantages, exactly these two aspects are emphasized in swarm-based models. These large-scale multi-agent models typically support dynamic interaction topologies, allow the agents to interact spatially, and they target the transition between local interactions and emergent global effects. The great variability in swarms not only demands for special diligence to maintain computational efficiency,

for instance by reducing the search space for interacting individuals based on preceding simulation states [586]. It also exalts the hardship of formulating and parameterizing the agents' behaviours—even the execution order of location update and velocity integration in simple flocking simulations yields fundamentally different global results [587]. These seemingly two distinct problems can both be tackled by optimizing the behaviours of swarm individuals.

In this article, we present selected works that show how swarms can be optimized to retrace global effects on the one hand and how they can be optimized to maintain computational efficiency on the other hand. In particular, the remainder of this article is structured as follows. In Section 17.2 we give a brief overview of select topics around the optimization of swarms (as opposed to using swarms for the purpose of optimization). Section 17.3 demonstrates how swarms can be adapted to meet specific expectations. In Section 17.4 we present an approach how swarm simulations could re-organize themselves during runtime to maintain computational efficiency. We conclude with a summary of this article and an integrative outlook on swarm optimization in Section 17.5.

17.2 Related Work

The work presented in this article is inspired and motivated by several disciplines of computer science and their applications. Craig Reynolds raised a lot of excitement in the computer graphics community when he demonstrated the simulation of flocking bird-oids, or boids, at the SIGGRAPH conference in 1987 [399]. Simple acceleration urges steered the boids in accordance with their local neighbourhoods through three-dimensionally rendered virtual worlds. The principles of large numbers of particles attracting and repelling one another in spatial simulations have also received considerable attention by physicists [95, 588, 94]. In many occasions, Eric Bonabeau, Scott Camazine and their colleagues built computational swarm models to retrace the biological behaviours of social insects [229, 230]. Marco Dorigo, James Kennedy their colleagues were forerunners to apply computational swarms for the purpose of optimization [589, 590].

17.2.1 Evolution of Constructive Swarms

Some of the mentioned scientists emphasized the applications of computational swarms for visualization or optimization, others focussed their efforts on the design of accurate biological models. Bonabeau et al. for instance, designed agent-based models to examine the nature of the cooperation of social insects. In models of nest construction, agents deposit particles triggered by environmental stimuli. Their behaviour was expressed in sets of rules that test the individuals' neighbourhood situations. Randomly chosen behavioural rules do not yield interesting structures. However, the researchers found rule sets that recreated the shapes of the different wasp genera's nests; Epipona, Parabolybia, Stelopolybia, Vespa, Chatergus. Marcin Pilat later added rule sets for the wasp families Agelaia, Parachartergus, and Vespula [591]. Motivated by the constructive character of these simulations, some of the authors of this article merged L-systems, formal production systems to generate plant-like geometric structures [338], with the interaction dynamics of swarms (swarm grammars, [343]). Similar to the work in which Henry Kwong and Christian Jacob interactively genetically bred novel parameter sets for boid flock formations [376], swarm grammars were also bred interactively and in immersive breeding grounds in three-dimensional space [344, 3].

17.2.2 Bottom-up and Cross-scale Modelling

Evolutionary breeding techniques have been used to optimize a vast range of computational models—from random boolean networks [592] and cellular automata [593] to L-systems [365] and membrane computing models [473]. Despite their algorithmic and formal universality, the underlying modelling approaches are designed to reflect special properties of the target systems; random boolean networks emphasize the interdependencies of genes, cellular automata and L-systems focus on fixed neighbourhood structures of differentiating cells, whereas membrane computing models, or p-systems, focus on the processes that occur between distinct tissues. Computational swarms find applications across scales—from molecular artificial chemistries to social science simulations—because of their inherently flexible interaction topologies and the focus on the relationship between local interactions and global effects. Therefore, Nelson Minar and his colleagues emphasized their multi-scalar properties and promoted a hierarchical design approach to swarm models [60].

17.2.3 Learning Hierarchies

First steps toward the design of emergent multi-scale models—where interactions on one level recursively determine the behaviour of the next higher levels, as opposed to chaining up differential equation systems that operate at different levels—were naturally taken in the domain of artificial chemistries. Rasmussen et al. designed a computational model in which increasingly complex structures emerge exhibiting novel properties—from monomers to polymers to micelles [10]. Although these experiments clearly retrace the formation of patterns at several levels of scale, Dorin and McCormack claim that such phenomena are not surprising given the model's simplicity. Dorin and McCormack argue that it takes considerably more effort to determine the novelties at higher levels in the hierarchy [264].

Dessalles and Phan foresaw a system in which detectors would identify emergent patterns in simulations and subsume the activity of the respective lower level objects [278]. Similarly, Denzinger and Hamdan introduced a modelling agent that observes the behaviours of other agents and maps them to predefined stereotypes [279]. Periodic re-evaluations of the agents' behaviours provided the opportunity to adjust the mappings in accordance with the dynamics of the system. Not only might the local interaction patterns change over time, but high-level phenomena might also influence the underlying layers. Lavelle et al. use the term immergence, or downward causation, to describe the impact of high-level organizations on entities at lower scales [594]. They postulate that explicit functions must be defined to bridge between micro and macro levels.

17.3 Guiding Emergence

Part of the fascination and the scientific value of computational simulation lies in the prediction of emergent phenomena. The driving computation may be based on various representations, e.g. mathematical equations, logical facts, or rule-based interactions. Numeric, iterative simulations can also be used to infer plausible underlying models for a given phenomenon, expressed by means of the utilized representation. Swarm-based simulations are of particular interest as they are typically setup to bridge the gap between local interactions and global, emergent

properties and processes[1]. In this section, we present several approaches to optimize the local behaviours of swarm individuals in order to retrace predefined emergent phenomena. Hereby, we rely on evolutionary computation techniques and we distinguish between fixed predefined target criteria and those that change over time.

17.3.1 Guiding along 2D Surfaces

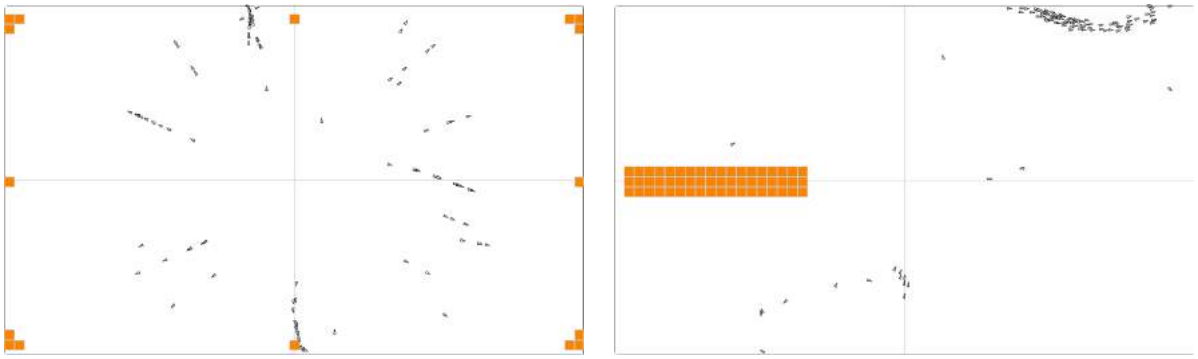
Inspired by observations of their natural counterparts, computational swarm models are often represented in two or three spatial dimensions. As the individuals' interactions depend on and impact the corresponding, spatially reflected interaction topologies, swarms lend themselves well for studying emergent phenomena that are graphically representable.

In [595], we showed how a virtual boid flock [1] can be bred so that its individuals maximize the time spent in predefined two-dimensional areas while flocking. In these experiments (Figs. 17.1 and 17.2), each swarm individual, or boid, is depicted as a triangle that is oriented towards its velocity. It identifies its neighbours inside of a forward-projected conic field of perception that is determined by a radius r and an angle α. To some extent, a boid accelerates randomly, however, its neighbours have a major impact on its trajectory. In particular, a boid follows an urge to align with its neighbours, to flock toward their geometric centre (cohesion urge), and to accelerate away from neighbours that are too close. This separation urge is triggered whenever a neighbour is closer than a given minimal distance. For the given experiment, the alignment, cohesion and separation vectors are normalized by dividing through the number of neighbours, whereas the random vector is normalized to a unit-vector. An individual's acceleration is computed by the weighted sum of these vectors. As a result, the genotype of a boid comprises the parameters for the field of perception (r and α), the minimum distance d_{min}, as well as the weight coefficients $c_{coh}, c_{sep}, c_{ali}, c_{ran}$ and limit values for both acceleration and velocity, a_{max} and v_{max}, respectively.

Figures 17.1 and 17.2 show boids that were optimized by means of an evolutionary algorithm to flock in the tiled areas. In Fig. 17.1(a) the flock breaks up into several clusters to reach the corners of the simulation space. In a second experiment, the flock formation shown in Fig. 17.1(b) achieves a high fitness value due to the great similarity between its shape and the

[1] Abduction refers to the corresponding logic-based approach to inferring the underlying parts of a model, whereas the field of inverse and ill-posed problems represents the mathematical, analytical analogue.

tiled target area. Another specimen that was discovered in the second evolutionary setting is presented in Figure 17.2. It solved the given, non-symmetrical task utilizing the constraints of the simulation environment, great dispersion but a great degree of connectivity among the boids. In Figure 17.2(a) the individuals spread radially from the origin. When repelled from the edges, the flock breaks into four parts (Fig. 17.2(b)). To the left and to the right, new clusters form and head back to the world centre (Fig. 17.2(c)), which makes at least one of the clusters pass across the tiles to the left centre of the simulation space (Fig. 17.2(c) and (d)).



(a) (b)

Figure 17.1: (a) A flock has learned to swarm to the edges of the simulation space. (b) The flight in formation of a broad stripe maximizes the flock's fitness when hitting the rectangular tiles.

In order to breed viable boid parameters for homogeneous flocks, we used a standard Genetic Algorithm (GA) which implemented: (1) Rank-based selection: 70% for the best 20%, 20% for genotypes between 20% and 50% of the ranks and 10% probability for the remainder of the parent population. (2) Recombination for half the offspring with multipoint crossover, normally distributed across the genotype. (3) Mutation of previous genotypes for the remaining offspring with a mutation probability $p = 0.2$ on single genes. We computed the phenotype fitnesses based on Equation 17.1. It sums the collisions of boids on all tiles, up to a maximum number of collisions per tile, over the course of a simulation. m denotes the number of swarm agents, n the number of tiles, t_{sim} the simulation time, and the function $c()$ yields the number of collisions between swarm individuals and an individual tile n_{ind} at time step t. In order to promote a smooth distribution of agents across the given tiles, the fitness evaluation function considers at most c_{max} agents on one tile. The final sum is normalized by the number of simulation steps and the number of swarm agents.

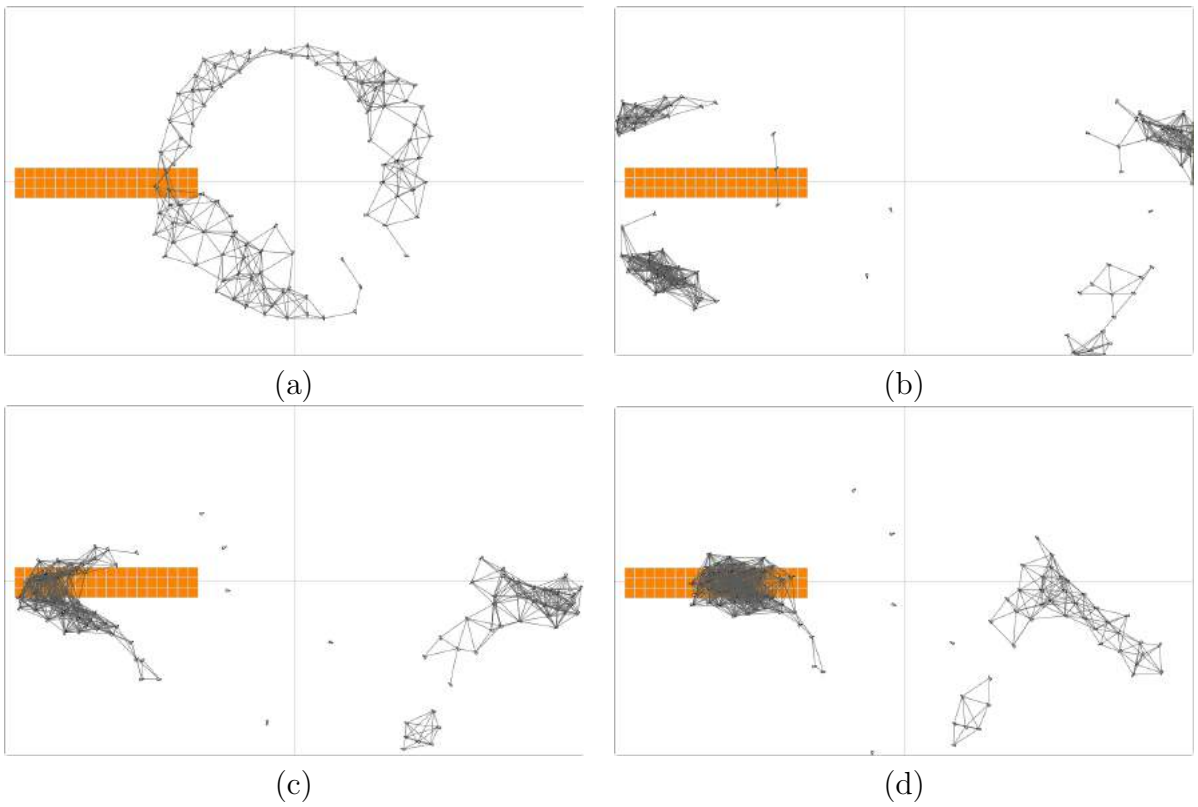


Figure 17.2: An evolved swarm relies on interactions with the environment in order to hit a non-symmetrical tiled area.

$$f_{2D} = \frac{1}{t_{sim} m} \sum_{n_{ind}=0}^{n} \sum_{t=0}^{t_{sim}} \min(c(n_{ind}, t), c_{max}), \ with \ c_{max} = \frac{m}{n} \tag{17.1}$$

The genotypes of the three presented cases are detailed in Table 17.1. The first one, depicted in Fig. 17.1(a), yields a high degree of scattered clusters due to the high cohesion and alignment weights and the narrow perception angle. The third genotype (Fig. 17.2) is a descendant of the second one (Fig. 17.1(a)). Its cohesion and alignment weights dropped significantly while its perception radius increased to the maximally possible value. d_{min} is greater than the actual perception radius in all three cases which implies that the separation urge was consistently triggered by all perceived neighbours.

17.3.2 Guiding through 3D Volumes

In [378, 596], we presented an approach to guiding swarm dynamics very similar to the one in Section 17.3.1. The model was inspired by work on nest construction in social insects

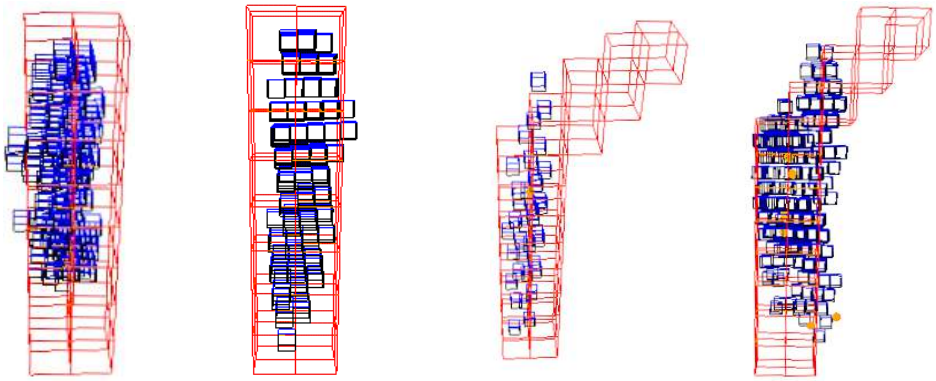
Phenotype	α	d_{min}	r	c_{coh}	c_{sep}	c_{ali}	c_{ran}	v_{max}	a_{max}
Figure 17.1(a)	0.74	90.28	56.16	4.23	1.62	5.0	0.55	6.51	20.02
Figure 17.1(b)	1.29	100.0	33.70	0.40	3.96	4.53	4.16	8.32	13.17
Figure 17.2	3.14	100.0	70.84	0.07	3.25	1.12	3.13	8.91	13.45

Table 17.1: Genotype vectors of the boid flocks shown in Figures 17.1 and 17.2 (rounded to two decimal places).

[229, 230]. In this model, in addition to following the flocking parameters outlined in Section 17.3.1, environmental stimuli prompted the individuals to place or remove cubic building blocks in virtual three-dimensional space (gravitation was not simulated, intersecting building blocks not allowed). The individuals' construction behaviour was expressed as *if-then* rules. The rules' antecedents would test the existence of up to five building blocks that were positioned relative to the acting individual. The consequence of each of twenty allowed rules could trigger the creation or destruction of a building block at specified relative coordinates, or it could set or reset the acting individual's point of focus coordinates. If set, the individual would be accelerated towards the point of focus alongside of the basic boid urges of alignment, separation, cohesion and some random acceleration. In addition, we also introduced an acceleration urge toward the ground that would increase with an individual's height. c_{foc} and c_{gro} denote the weight coefficients for these two additional urges, respectively.

Again, we used a standard GA to breed swarms that were guided by geometrical constraints. This time, the fitness of a swarm was determined by the ratio of building blocks built inside and outside of a predefined three-dimensional structure composed of a set of cubes. An initial seed cube marked the site a swarm's construction efforts would be measured against. Figure 17.3 shows the predefined structures and the swarm-based constructions of two different experiments. Instead of multi-point crossover operators, recombination is performed for 40% of the offspring based on a randomly generated two-point crossover mask that preserves pairs of dependent rules with a greater probability. The number of rules of the offspring is limited to the smaller number of rules of the parents. The parents for all the offspring were chosen by means of fitness proportionate selection. In addition the ten best individuals were always considered as parents (kBest with $k = 10$). Mutation is performed per boid gene with a probability of $m_{boid} = 0.2$, whereas the conditions, the action, and the action parameter (a relative position) are considered for mutation independently with a probability $m_{rule} = 0.1$. In the evolutionary experiments, we emphasized the coordination of construction and fixed some of the flocking

parameters. In particular, $d_{min} = r = 2.0$, $\alpha = 2.0$, $v_{max} = 0.5$, and $a_{max} = 0.3$. Please note that for these experiments a different simulation environment, VIGO [597], was used which resulted in a spatial scaling factor much smaller than in Section 17.3.1.



(a) (b) (c) (d)

Figure 17.3: Swarm constructions (inner aggregations) are guided by predefined 3D structures (outer grids).

The construction rule sets of the two independently bred swarms depicted in Figure 17.3 were dominated by unconditional and conditional rules for cube creation. Each of the swarms also set and reset the individuals' focusses (3 unconditional construction rules in (a-b), 4 in (c-d), and 2 conditional ones in both specimens). In the swarm depicted in Figure 17.3(a-b), the individuals also unconditionally removed construction elements in a relative location. Further information about these rule sets can be found in [596].

Phenotype	c_{coh}	c_{sep}	c_{ali}	c_{ran}	c_{foc}	c_{gro}
Figure 17.3(a),(b)	0.18	0.06	0.30	0.00	0.14	0.17
Figure 17.3(c),(d)	0.16	0.43	0.16	0.00	0.23	0.12

Table 17.2: Flocking genotypes of the constructive swarms shown in Figures 17.3(a),(b) and (c),(d), respectively (rounded to two decimal places).

17.3.3 Tracing and Learning Flock Dynamics

The speciality of a swarm is its inherently dynamic interaction topology and the resulting feedback on its global behaviour. In [117], we analyzed previously discovered boid flock specimens [376] based on their interaction topologies over time. We also presented an approach to finding new flock configurations whose interaction topologies evolved in accordance with predefined

functions that reflect naturally occurring phenomena such as biological switches and clocks or timers. In particular, we showed that a step function can be approximated by a flock's average neighbourhood degree $\bar{n}$, if its individuals slowly drift away from one another and that an oscillating neighbourhood degree can be established by a pulsating flock. Here, we want to share the latter example, as its characteristic sequence of phase transitions is especially interesting in the context of complex simulation research.

As the initial configuration of a complex system may heavily impact the results of a numeric experiment, we encoded the initial configurations (position, velocity and acceleration) of individuals as part of a swarm's genotype, similar to an epigenetic factor. In order to provide a spatial point of reference, we allow the swarm to urge toward the world centre, $\boldsymbol{o} = (0, 0, 0)^T$ (weighted by c_{foc}). This time, we simply configured a Genetic Algorithm with fitness proportionate selection, incremental mutation and multi-point crossover on all numeric values. To enforce the approximation of a predefined target function, we computed the following fitness value:$f_{oscillation} = 1/\left(\sum_{t=1}^{40} |\bar{n}(t) - x(t)|\right)$. Over a period of 40 time steps, the fitness diminishes proportional to the absolute difference between the target function $x(t)$ and its approximation $\bar{n}(t)$.

Figure 17.4 shows the neighbourhood function $\bar{n}(t)$ as exhibited by the evolved swarm configuration listed in Table 17.3. The oscillation happens as the flock repeatedly expands (Figure 17.5) and contracts (Figure 17.6). Leaps from a plateau to a local maximum, as seen at $t = 100$, occur when formerly separated flocks rejoin. Eventually, at $t = 1244$, the oscillation ends (Figure 17.4(b)); this is when the agents form a tight cluster and start orbiting around the world centre. In order to facilitate the identification of flocking patterns, we activated motion blurring in the renderings.

Phenotype	α	d_{min}	r	c_{coh}	c_{sep}	c_{ali}	c_{ran}	c_{foc}	a_{max}	v_{max}
Figures 17.5 & 17.6	2.64	4.12	7.86	0.95	0.53	0.76	0.76	0.36	12.15	7.16

Table 17.3: Evolved swarm parameters that result in the neighbourhood evolution shown in Figure 17.4. The corresponding flocks oscillate though repeated contraction and expansion (Figure 17.5).

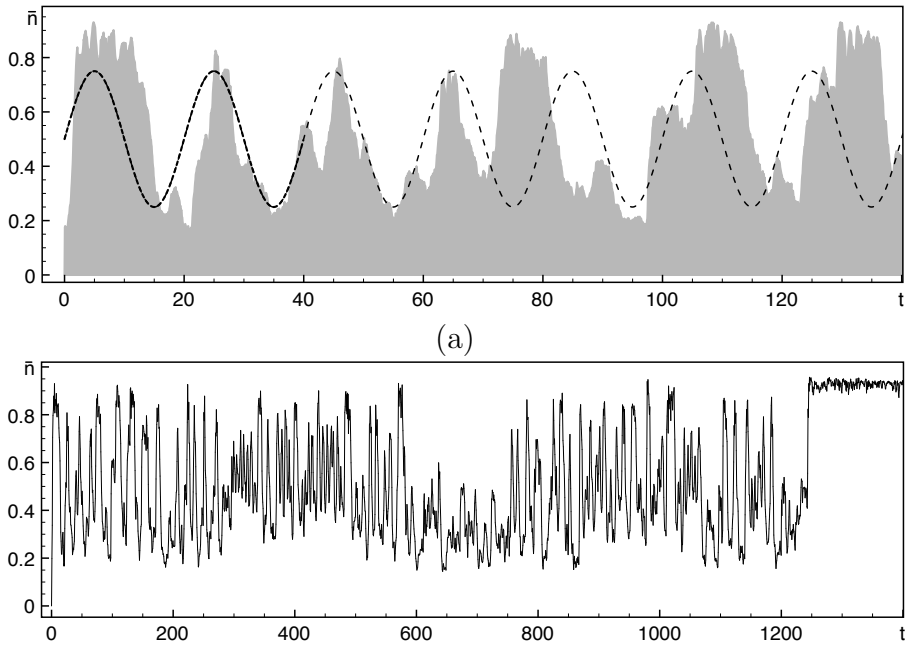


Figure 17.4: (a) The average neighbourhood of a flock $\bar{n}$ approximates a sine function that it learned until $t = 40$. (b) At $t = 1244$, the flock forms a tight cluster and remains in an equilibrium with $\bar{n} \in [0.35; 0.45]$.

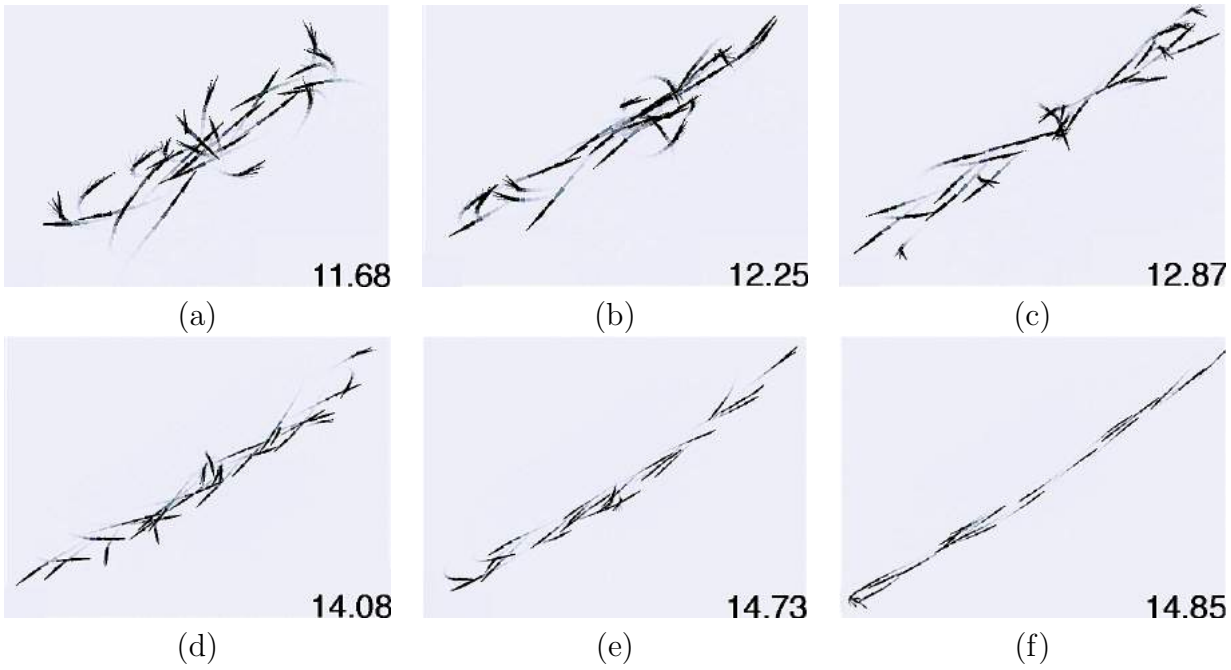
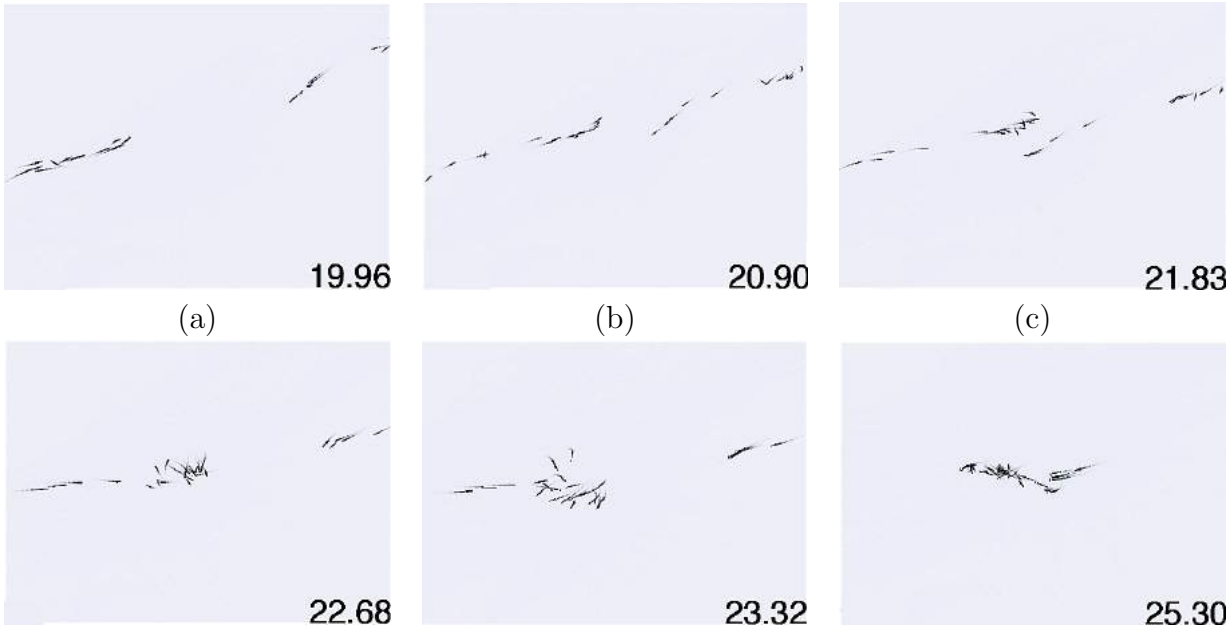


Figure 17.5: The series of images shows how a swarm in a knot formation expands to two sides. Eventually, two flocks emerge and head into opposing directions.



(a) (b) (c)

(d) (e) (f)

Figure 17.6: (a-b) Two flocks head toward the world centre from opposing directions. (c) They avoid each other at first. But soon they closely interact again. (The images were adjusted to fit both flocks, the zoom was slightly increased once for capturing (d-f)).

17.3.4 Parameter Optimization in a Heterogeneous Predator-Prey Model

As a test bed for learning the behavioural parameters of heterogeneous swarms, we chose a classic predator-prey model, in which the populations of prey p and predator individuals P depend on one another [224, 225]. The Lotka-Volterra differential equations (DEs) describe the dynamics of a predator-prey ecosystem (Equations 17.2 and 17.3). In our corresponding, two-dimensional swarm-based model, both prey and predators wander about randomly. Prey dies when encountering a predator. It also dies of other causes with probability β at each step of the simulation, or it reproduces with probability α. Predators prosper from nutritional encounters with prey individuals and reproduce on the spot with a probability γ. Their deaths occur with probability δ. The populations of predator and prey individuals, p_{init} and P_{init}, are initially set to magnitudes between 10 and 500.

$$\frac{dp}{dt} = p(\alpha - \beta P) \tag{17.2}$$

$$\frac{dP}{dt} = -P(\gamma p - \delta P) \tag{17.3}$$

We reverse engineered the parameters for the swarm model relying on several algorithms. First, we discretized the continuous results of Equations 17.2 and 17.3 by means of an online time-series segmentation algorithm [598]. We then measured the similarity value between the time series produced by the swarm-based model and the segmented differential equation results using a generic Dynamic Time Warping Algorithm [599, 600]. This measure served as the fitness value to search for adequate solutions based on particle swarm optimization (PSO) [226].

Different from the experiments presented in Section 17.3.3, the swarm individuals in this predator-prey model cannot alternate their velocities. Therefore, in order to approximate a given plot with a fixed time scale, we optimized for qualitative similarity between the swarm simulation and the DE-system. We accomplished this by adding the number of simulated steps to the swarm configuration. A single scalar factor suffices to match the evolved and the expected graphs.

In order to foster robust solutions, we ran each simulation three times for a given set of parameters and considered the average performance as the particular swarm's fitness value. Twenty optimization experiments yielded two prototypical swarm configurations (Table 17.4). Their average evolution over the course of one simulation is depicted in Figure 17.7. While the overall PSO experiments have converged on two different solutions, each of them is still close to the DE-based results. The second class of solutions, Figure 17.7(b), qualitatively matches the DE model better as the population of prey individuals recovers at the end of the simulation. We give credit for this development to the greater reproduction rate α of prey individuals as seen in Table 17.4. The shift between the swarm-based approximations and the DE-based target graphs in Figure 17.7 is the result of a relatively generous error threshold for the similarity measures.

Phenotype	α	β	γ	δ	$\lvert p_{init} \rvert$	$\lvert P_{init} \rvert$	steps
Figure 17.7(a)	0.38	0.13	0.64	0.18	432.63	317.13	132.63
Figure 17.7(b)	0.76	0.30	0.69	0.20	436.44	330.64	119.26

Table 17.4: Average parameters of two classes of swarm-based predator-prey models that were found using particle swarm optimization (rounded to two decimal places).

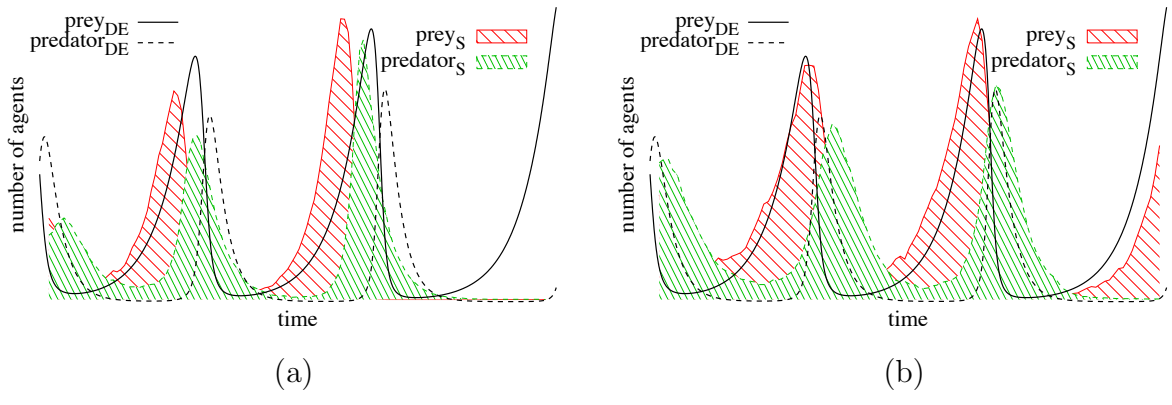


Figure 17.7: (a) and (b) show the population dynamics of two prototypical swarm configurations compared to the results of the Lotka-Volterra DE model of a predator-prey system. The results of both modelling approaches had to be scaled to match (see *steps* in Table 17.4), yielding these qualitative diagrams.

17.4 Abstract and Scale

In the previous section, we demonstrated the optimization of swarm behaviours in respect to statically measurable outcomes, dynamics over time, and heterogeneous system compositions. While the resulting systems may suffice to retrace and explore certain isolated phenomena, the extensibility of swarms, their intrinsic potential to interface with newly introduced elements and to yield high-level emergent properties renders scalability of swarms another great challenge.

The flexibility of swarm-based modelling comes at a cost. Without further optimization, the identification of interaction partners of n swarm individuals alone yields a computational complexity of $O(n^2)$. Typically, the interaction scope of large numbers of units may, therefore, be drastically reduced. The interaction in spatial environments are often limited to to the local, discrete neighbourhoods relying on discrete computational modelling approaches such as cellular automata or cellular potts [601]. However, the ability of the models to continuously change their interaction topologies among the agents is crucial to capture the systems' dynamics responsible, for instance, for emergent transportation [94]. Of course, this confinement does not only apply to spatial interactions but to the number of dimensions of interactions in general, to the number of individual interaction rules, and to the number of simulated individuals. A system of automated abstraction, which learns the local patterns and subsumes them as high-level agents, offers a perspective for a truly scalable computational approach. Instead of learning behaviours motivated by predefined patterns as exercised in the previous section, we now demonstrate how emergent patterns that occur among (properly trained) agents can be learned, rephrased as higher level behaviours, and utilized to reduce the number of simulated agents.

17.4.1 Towards Self-organizing Hierarchies

Early on when we started our investigations, we already had a rather clear picture of our envisioned abstraction framework. It should automatically, and in a decentralized fashion, create abstractions in a simulation, whenever possible, and abolish them, whenever necessary. As we imagined it to be primarily deployed in swarm systems, it was obvious to us that the abstractions should be discovered and managed by special swarm individuals that are immersed

into arbitrary swarm simulations. We termed this concept self-organized middle-out abstraction approach, or SOMO [268]—"middle-out" referring to the idea that it would create higher level representations (bottom-up) but also break them down again (top-down).

However, in order to ensure the validity of our conceptual foundation, we narrowed down the scope of our first set of experiments [9]. Therein, we identified correlated nodes in gene regulation networks (modelled by a set of simple differential equations), approximated their behaviours as groups by means of artificial neural networks (ANNs) and subsumed the lower level nodes by high-level agents, or meta-agents. High correlation values between concentrations would consistently yield higher level agents, whereas drops in the correlation values of previously grouped nodes resulted in the removal of the respective, outdated abstraction. Figures 17.8(a-b) and (e-f) show the results of this *greedy* approach when applied to two different MAPK pathway models, one resulting in a sigmoidal concentration of the MAPK-PP protein [7], the other one in a periodic expression pattern [8]. The relationship between inaccurate emulation by the meta-agents and the number of meta-agents in the system is obvious when comparing Figures 17.8(a) and (e): The occurrence of dips in the otherwise smooth approximative graph triggers the removal of abstractions. In the periodic model, changes occur too frequently to be accommodated by the meta-agents which resulted in a high frequency of their creation and removal (Figure 17.8(d)).

Although the overall performance of the the greedy abstraction approach was far from satisfactory, it successfully reduced the number of agents in the system. In our second set of experiments, we attempted to amend the particularly short lifespans of the abstractions seen in the periodic MAPK model in Figure 17.8(d). So we promoted a dynamic management of the learned meta-agent hierarchies [247]. Whenever a meta-agent became obsolete, it would restore the subsumed, previously active abstraction hierarchy. Figures 17.8(c-f) depict the results of this *hierarchical* approach. The stepwise restoration of lower-level abstractions is clearly identifiable in Figure 17.8(e): At about $t = 2250$ one meta-agent, which was trained by means of standard Genetic Programming (GP), is removed and its two underlying meta-agents are re-introduced into the simulation. Before this point in time, the learning process consistently built greater abstractions. The predictions by the meta-agents exhibited greater inaccuracy than in the greedy case. In addition, the divergence between target graph and approximative results (Figure 17.8(c)) does not coincide well with the creation and removal of meta-agents

(Figure 17.8(e)); it is surprising that yet another hierarchical level is added shortly after time step $t = 2000$, even though the preceding emulated concentration strongly deviated from the target function.

17.4.2 Immersive Decentralized Abstraction

We believe that the optimization and further the situation-dependent choice of apt parameter set for the efficient abstraction and hierarchy management necessitate in-depth studies on top of a fully fledged SOMO prototype. Therefore, for our next experiments, instead of fine-tuning the parameters to optimize the ratio between agent-reduction and accurate emulation, we searched for a better learning example—one that allows for the deployment of self-organizing, abstracting swarm individuals in the context of a swarm simulation. As previous results had suggested (Section 17.4.1), linear instead of periodic system dynamics promised the best results for a prototype SOMO implementation. Hence, we adjusted the SOMO system and designed swarm individuals that could be immersed into a swarm simulation of the physiological process of blood coagulation.

In addition to the swarm individuals of the model, or *model agents*, we designed an *observer agent*. It observes model agents and logs their interactions in an *interaction history* that serves as a database for pattern recognition. An entry in an interaction history contains, for instance, a reference to the acting agent A, the executed action *act* with time stamp t along with the set of interaction partners $\mathcal{A}$. In our prototype, the observer applies a k-means clustering algorithm [602] to find a cluster of overlapping interaction partners as soon as the interaction history contains a sufficiently large set of logged entries. Once a cluster is identified, the observer infers a generalized group behaviour from the logged interaction data: It learns the information that remains fixed across the set of relevant rules and it identifies boundaries, periodicities and probabilities of reoccurring variable actions. All the logged interactions that led to the rules of the newly phrased group behaviour are removed from the lower level individuals and the observer starts performing on their behalf. Initially, the observer has an unbiased confidence in a newly learned abstraction. Periodically testing the behaviour of the subsumed agents in the current situation lets the observer adjust this confidence—it grows, if the predictions were correct, otherwise it drops. The observer fully restores the subsumed agents, should the confidence drop below a certain threshold. The behavioural subsumption by the observer reduces

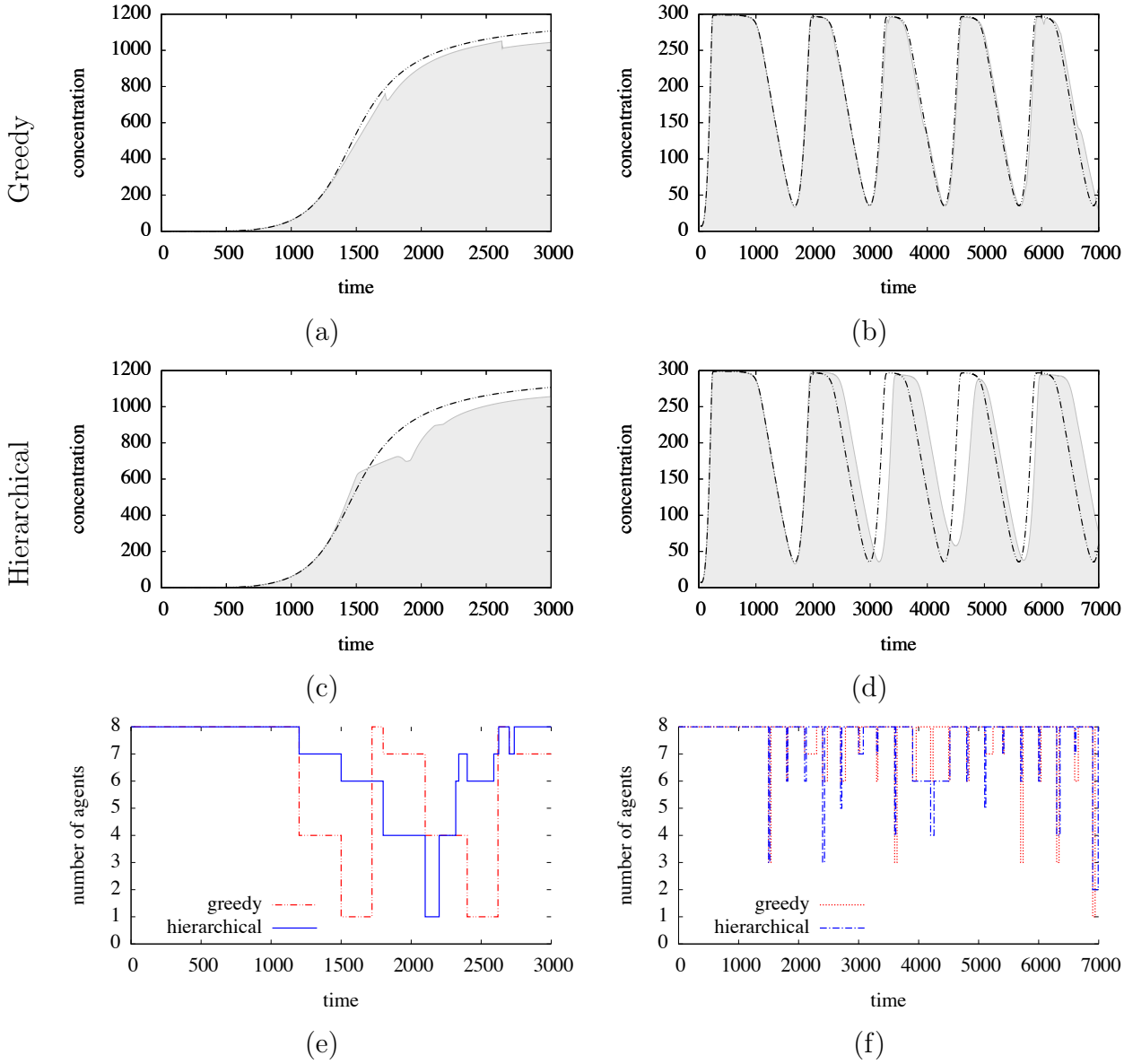


Figure 17.8: Simulations based on a non-periodic and a periodic MAPK pathway model are shown on the left-hand and the right-hand side, respectively. Comparisons between the differential equation system and a greedy (a-b) and a hierarchical (c-d) abstraction approach are shown. The results of the DE model are indicted by dashed lines, the agent-based dynamics are depicted as shaded areas. The numbers of agents deployed by the abstraction approaches are compared in (e) and (f) (individual legends are provided in these two diagrams).

the otherwise necessary tests for triggering actions and the search for the respective interaction partners. Of course, in a deployment scenario, this performance gain would be measured against the computational overhead for observing model agents, abstracting, validating, and possibly removing group behaviours.

We immersed the prototype SOMO observer in a swarm-based blood coagulation simulation in which bio-agents aggregate at a wound site and form a clot (Figure 17.9). After $t = 100$, the observer identifies $k = 30$ clusters in its interaction history and the centroid of the largest cluster is considered to be the learned group behaviour for which $[t_{min}, t_{max}]$ and p_{exec} are inferred. At intervals of $\Delta = 10$, the observer updates its confidence values; abstractions with confidence values below $\tau = 30\%$ are revoked. In this environment, our prototype successfully identified and abstracted behaviours such as random movement, executed with probability $p_{exec} = 100\%$ and $t \in [0, \infty]$, and state changes induced by collision ($p_{exec} = 77\%$ and $t \in [90, 95]$). Due to the model's simplicity, the number of calculated situations over the course of a simulation increases linearly with the number of incoming bio-agents (introduced by the blood stream). Our prototype managed to keep this number constant (Figure 17.10). Its overhead is shown in the additionally computed situations that occur just before the abstraction starts ($t < 100$). The peaks in our proposed method indicate the intervals at which some model agents are allowed to execute their actions.

17.5 Summary and Future Work

Swarm-based models and simulations bridge the gap between the level of local interactions and global system behaviours. Instead of programming a swarm system, one has to program its individuals, and in such a way that the whole swarm can accomplish its task. A computational swarm might, for instance, be designed to retrace and predict natural phenomena, to optimize mathematically phrased problems, or to support creative design decisions. In this article, we presented several experiments that elucidate how the behaviour of swarm individuals can be programmed.

First, in Section 17.3, we focussed on the interplay of globally defined constraints and the inferred behaviours of locally interacting swarm individuals. Due to the spatial properties of basic boid swarms, we formulated tasks geometrically to (1) evolve flocking swarms in 2D and (2)

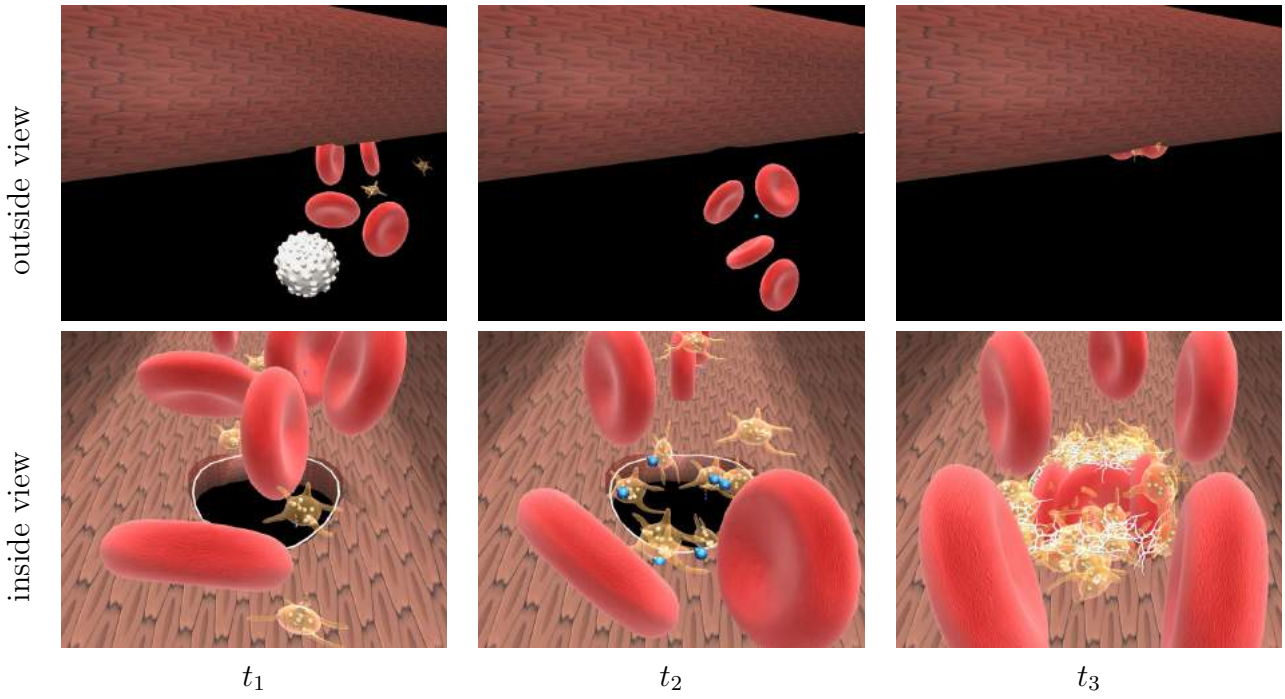


Figure 17.9: A swarm-based blood coagulation simulation shown from two perspectives at three consecutive time steps $t_1 ... t_3$.

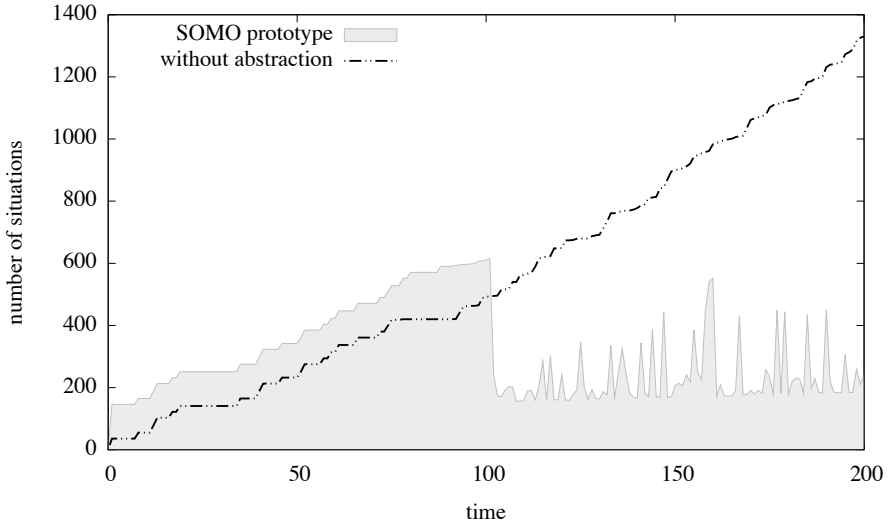


Figure 17.10: Number of agent situation evaluations over the course of a blood coagulation simulation, with and without SOMO abstraction.

constructive swarms in 3D (by means of Genetic Algorithms). (3) We introduced a quantitative measure to capture the neighbourhood dynamics of boid flocks that allowed us to genetically breed boid individuals that would, in a group, approximate a predefined neighbourhood density function. (4) A heterogeneous swarm model of predator and prey concluded our explorations of guiding emergence; here, a system of differential equations specified the system dynamics, and the parameters of the two types of swarm individuals were learned (by means of particle swarm optimization).

In Section 17.4, we then presented several stages toward an inherently scalable approach to swarm simulation, the Self-organized Middle-out abstraction framework, or SOMO. Here, meta-agents subsume the behaviours of lower level individuals based on re-occurring interaction patterns in order to reduce the number of required computation steps. Meta-agents organize themselves in hierarchies that are dynamically built up and broken down, depending on the demands of the ongoing simulation and the predictive power of the learned abstractions. In our experiments, we first (5) greedily subsumed low-level agents by meta-agents in an easily verifiable differential equation model of the MAPK signalling pathway (mitogen-activated protein kinase). (6) We introduced a dynamic management of hierarchies so that, upon the identification of an obsolete abstraction a preceding abstraction is restored instead of resetting all the learned accomplishments all at once. Finally, we (7) equipped special swarm individuals, so-called observer agents, with a behaviour to build and manage abstraction hierarchies based on interaction histories of groups of monitored individuals.

While examples (1) to (4) emphasize the top-down learning, breeding, or optimization of the behaviour of swarm individuals, instances (5) to (7) attempt the opposite; the SOMO approach learns and utilizes patterns that emerge from local interactions bottom-up, only breaking them down again should it become necessary. As much as these perspectives might differ, we believe, that they might serve as forerunners of an algorithmic framework for integrative, large-scale and multi-scale modelling and simulation. In the last paragraph of this article, we want to outline how this could work, at the same time implying a suggested direction of future work in this field.

The more specialized the interaction patterns a SOMO observer is looking for, the more efficiently will it identify and abstract them. A set of differently configured SOMO observers spread across the simulation space could evolve based on their success to abstract in their re-

spective niches—one may assume that activity is strongly heterogeneous across the interaction dimensions of most large-scale simulations. At this point, the unsupervised online learning process of SOMO would be two-tear, considering the accuracy of the generated abstraction hierarchies and the configuration of the observer agents. Additional top-down constraints could be introduced by a second observer type that reconfigures individuals in order to reproduce specific process patterns. Such a top-down observer could substantially change the original model, so its influence should be strictly constrained. The conditional introduction and removal of top-down observers, depending, for instance, on the emergence of certain high-level behaviours learned by the currently implemented bottom-up SOMO observers, would enable an external modeller to embed expected milestones into a bottom-up computed multi-scale simulation and ensure the seamless computational integration of its scales.

Chapter 18

Abstraction of Agent Interaction Processes: Towards Large-Scale Multi-agent Models

The typically large numbers of interaction in agent-based simulations come at considerable computational costs. In this article, we present an approach to reduce the number of interactions based on behavioural patterns that recur during runtime. We employ machine learning techniques to abstract the behaviour of groups of agents to cut down computational complexity while preserving the inherent flexibility of agent-based models. The learned abstractions, which subsume the underlying model agents' interactions, are constantly tested for their validity—after all, the dynamics of a system may change over time to such an extent that previously learned patterns would not reoccur. An invalid abstraction is, therefore, removed again from the system. The creation and removal of abstractions continues throughout the course of a simulation in order to ensure an adequate adaptation to the system dynamics. Experimental results on biological agent-based simulations show that our proposed approach can successfully reduce the computational complexity during the simulation while maintaining the freedom of arbitrary interactions.

Abbas Sarraf Shirazi, **Sebastian von Mammen**, and Christian Jacob. *Abstraction of agent interaction processes: Towards large-scale multi-agent models.* Simulation 89 (2013), no. 4, pp. 524–538.

18.1 Introduction

Phase transitions in complex systems cannot be inferred from the properties of the underlying parts. Rather they occur due to the interactions of the involved variables [278]. The agent-based modelling approach is a well-suited means to model complex systems, as it provides each part of the system with the ability to change its own state and to interact with other parts. Agent-based computational models have also gained great popularity as they can address heterogenous populations, noise, spatial and temporal relationships [304, 603, 604, 241].

The flexibility of agent-based models renders their simulation computationally inefficient [605]. As each agent could potentially interact with all the other n agents, merely identifying who interacts with whom becomes a computationally expensive task—$O(n^2)$ in the worst case. To overcome this problem, agent-based simulations are often limited to fixed neighbourhoods in discrete lattice spaces as implemented by cellular automata [606, 246, 607]. However, the ability of a model to continuously change the interaction topology among the agents is crucial to trace, for instance, the dynamics of transportation effects [94] or developmental processes [342].

In this article, we present an approach to apply machine learning techniques such as evolutionary algorithms, neural networks, and clustering in order to reduce the computational costs of an agent-based simulation while preserving its inherent flexibility. In particular, we show how groups of agents that exhibit behavioural patterns can be reduced to single agents with (computationally) simplified interaction rules. In order to identify a group of agents that can be substituted by a single agent, either neighbouring agents form a group, or, more generically, an *observer agent* monitors arbitrary groups of agents and substitutes them based on their exhibited behavioural patterns. As the agents' interactions may vary over time, the learned behavioural patterns may loose their validity. Therefore, confidence values determine the lifespan of the learned behavioural abstractions. Continuous re-evaluation of these confidence values allows for a self-organized optimization process in which the substitutions are adaptively created and revoked.

The remainder of this paper is organized as follows. Section 18.2 reviews related works in multi-agent modeling and abstraction. In Section 18.3, we first show how we can use artificial neural networks to learn the collective behaviour of agent groups. Next, we present an approach

that relies on genetic programming and manages agent hierarchies dynamically, i.e. it does not destroy the learned abstractions completely should their confidence values drop, but only incrementally, as needed. In this context, we also elucidate the algorithm that ensures the validity of any learned patterns. Section 18.4 further refines the introduced approaches to consider arbitrary types of agent interactions including collisions and state changes. In order to demonstrate the effectiveness of this refined approach, we apply it to an agent-based blood coagulation simulation. Finally, concluding remarks are presented in Section 18.5.

18.2 Related Work

Abstract knowledge represents higher-order patterns that occur in lower-level concepts. It bears the essence of a system and ignores unnecessary details [608, 609]. Higher-order patterns emerge from the interactions of the parts, or agents, of a system [604]. In natural systems the formation of higher-order patterns happens across several scales of time and space, which renders their complete description impossible. However, it has been suggested that one could approximate the multiple scales of natural systems and their interdependencies by means of computational models that incorporate hierarchies of agents. High level agents in such hierarchies correspond to high degrees of abstraction of the system processes. In this section, we briefly describe some of the related works that motivated or addressed this concept.

18.2.1 From Bottom-up to Abstract Models

Artificial chemistries [337] and computational developmental systems—such as L-systems [338], relational growth grammars [176], or swarm grammars [3]—explicitly, often visually trace the emergence of high level structures based on simple constituents. These constituents may be represented as formal symbols or as entities in physics simulations. Complex interaction patterns can emerge from even the most simple interactions. Autocatalytic networks, for example, denote patterns of chemical reactions that nurture one another [581]. Stable interaction networks may even exhibit the property of self-replication [277]. As a result, the formation of intertwined entities is promoted and hierarchies of increasing complexity emerge in nature [261].

Rasmussen *et al.* designed a computational model based on artificial chemistries, in which

structures are formed with an increase in complexity and with different functionalities—from monomers to polymers to micelles [10]. Although these experiments clearly retrace the formation of patterns at several levels of scale, Dorin and McCormack claim that such phenomena are not surprising given the model's simplicity. Dorin and McCormack argue that it takes considerably more effort to determine the novelties at higher levels in the hierarchy [264].

A first step toward the identification of high-level patterns is to gain clarity about the abstractions inherent in an agent model to begin with. Bosse *et al.* propose the classification of types and levels of abstraction of agent-based models based on the following dimensions [610]:

The Process Abstraction Dimension deals with the behaviour representation of an agent, e.g. whether an agent is modelled by its inputs and outputs, whether other variables like beliefs or desires are also considered, or whether even lower level properties of an agent are modelled.

The Temporal Dimension deals with the definition of the agents' behaviours over smaller or longer periods of time.

The Agent Cluster Dimension specifies the granularity of the agent-based model, i.e whether an individual agent represents an entity or a cluster of entities.

Ralambondrainy *et al.* identify the complex task of observing a simulation, for which they propose a separate multi-agent system [611]. They describe an ontology to facilitate the communications of the agents in the second system. The observation agents have three main tasks, namely (1) acquisition of observational elements, (2) processing of simulation results, and (3) presentation of the results to human actors. Although the second system does not affect the original simulation, the notion of a separate system with the ability to present higher-level, abstract knowledge emphasizes the necessity to have external observers in the simulation.

Several approaches rely on a priori definitions to identify emergent patterns in agent-based simulations. Servat *et al.* acknowledge that simulation states can provide clues for the introduction and configuration of high-order agents [612]. However, they insist on the necessity to predefine the behaviours of high-level agents. The same is true for Chen *et al.*'s formalism which they specifically use for validating predicted behaviours [613, 603].

In order to capture emergent phenomena, Dessalles and Phan foresaw a system in which detectors would identify emergent patterns and subsume the activity of the respective lower level objects [278]. Similarly, Denzinger and Hamdan introduce a modeling agent that observes perceivable behaviours of other agents and maps them to a predefined stereotype [279]. But Denzinger and Hamdan also present a novel aspect: The periodic re-evaluation of the agents' behaviours gives the modeling agent the opportunity to adjust the mappings in accordance with the dynamics of the system. Not only might the local interaction patterns change over time, but high-level phenomena might also influence the underlying layers. Lavelle *et al.* use the term *immergence*, or downward causation, to describe the impact of high-level organizations on entities at lower scales [594]. They postulate that explicit functions must be defined to bridge between micro and macro levels.

Cardon proposes three organizational levels to control the behaviour of a multi-agent system [614]. The constituent agents are defined in the aspectual level. A geometrical mapping of aspectual agents forms the second level called the morphological level. Using a simplified, higher-level representation of agents in the morphological level, analysis agents in the evocation level identify the current state of the simulation and control the agents in the aspectual level by tampering their behaviour. This approach provides self-adaptability in the system while enforcing a degree of control on the behaviour of the system as a whole.

von Mammen *et al.* introduced the concept of *self-organized middle-out abstraction (SOMO)*, where observer agents monitor the interaction history of sets of agents, use motif discovery to detect recurrent patterns, and create hierarchies of high-level agents that subsume the lower interacting agencies [268]. Although they do not exclude the possibility of a relationship between learned high-order patterns and emergent phenomena found in nature, SOMO primarily targets an increase of efficiency by repeatedly substituting groups of agents by individual high-level instances that work at lower computational costs.

The authors of this article have previously demonstrated that high-level agent substitution indeed results in a reduction of computational costs [9, 247]. In particular, we deployed artificial neural networks and genetic programming, two established inductive learning methods, to learn agent abstractions in a model of a biological signaling pathway. Clusters of biological substrates and their corresponding activation patterns were substituted by meta-agents. We recently extended our earlier implementation by introducing observer agents that are able to abstract

arbitrary patterns of groups of agents [605].

18.2.2 Toward a Framework for Multi-scale Modeling

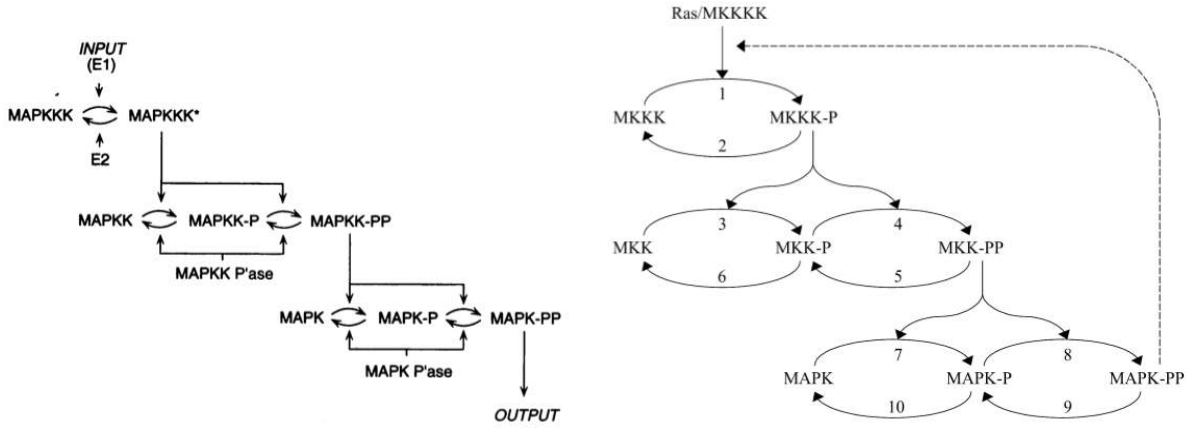
As technology advances the design of multi-scale models becomes more prominent. As long as these approaches merely connect models of different scales and feed back and forth the computed results as variable parameters, the challenge can be addressed with the right level of domain knowledge and software engineering skills [615, 87]. As discussed in Section 18.2.1, there are only few concepts that address the issue of automatic identification and abstraction of emergent patterns, which is crucial for a system that would identify new levels as a result of the computational process.

Martins *et al.* review different multi-scale models (from biomolecules to cells, tissues and organs) and conclude that despite the lack of a quantitative model of a cell, such models may help understand cancer growth and its therapy [616]. Erson and Cavusoglu propose a software framework for multi-scale model integration and simulation [617]; however, no specific modeling technique is described. There are a few physical multi-scale models, e.g. CPM [618], and Synergetics [619]. However, as of yet, there is no universally adopted computational framework for the assembly of multi-scale biological models [301].

Bassingthwaighte *et al.* identify a systems approach for developing multi-scale models which includes six steps [620]: (1) the definition of the model at its highest level of resolution, (2) the abstraction of patterns ("reduced-form modules"), (3) the identification of valid parameter ranges of these abstractions, (4) the observation of the variables of the system, (5) replacement of higher resolution models with abstractions, and (6) the validation of the performance of the multi-scale model against available real-world data. The authors further discuss open challenges of their approach such as parameter identification in closed-loop systems and the identification of input-output delays.

18.3 Abstraction in the MAPK Signaling Pathways

A signaling pathway describes how information travels from the receptors of a cell to an inside target [245]. Typically, the information ripples through a cascade of biochemical reactions



(a) (b)

Figure 18.1: (a) The MAPK signaling pathway (from [7]), and (b) The MAPK signaling pathway with a negative feedback (from [8]).

that are carried out by enzymes. The Mitogen-Activated Protein Kinase (MAPK) pathway plays a key role in the cell cycle and is extensively documented. It is responsible for responses to extracellular stimuli and regulates cellular activities, such as gene expression, mitosis, and differentiation [246]. In the MAPK signaling pathway proposed in [7], a hypothetical enzyme E1 stimulates the cell and results in an increase in production of the MAPK-PP enzyme (Fig. 18.1(a)). In another model [8], a negative feedback loop causes sustained oscillations in the production of MAPK-PP (Fig. 18.1(b)).

The diagram in Figure 18.1 describes the interaction topology of substrates. Numerical differential equation solvers are used to calculate their concentration updates over the course of time. For example, the update formula for the MAPK-PP concentration is given as follows:

$$d[MAPK - PP]/dt = v_8 - v_9 \tag{18.1}$$

$$v_8 = \frac{k_8 \cdot [MKK - PP] \cdot [MAPK - P]}{K_8 + [MAPK - P]} \tag{18.2}$$

$$v_9 = \frac{V_9 \cdot [MAPK - PP]}{K_9 + [MAPK - PP]} \tag{18.3}$$

where k_8, K_8, V_9, and K_9 are constants and $[X]$ is the current concentration of substrate X. The complete set of update equations can be found in [7] and [8].

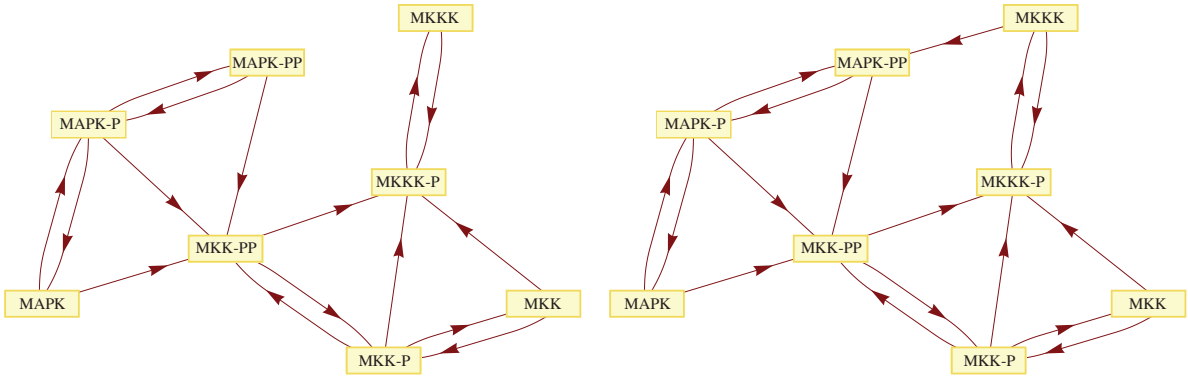


Figure 18.2: Agent interaction graphs for the MAPK signaling pathways of Fig. 18.1.

Amigoni and Schiaffonati present three approaches to multi-agent simulations of the MAPK pathway [245]. In the first approach, each chemical reaction is represented as an agent [246]. The second approach translates each intracellular component into an agent that uses a blackboard mechanism [621] to interact with other agents in the system [622]. In the third model, each molecular entity acts as an agent [623]. For our model, we follow the last approach and consider each substrate a loosely defined, independent agent. Their behaviours are determined by the interaction graphs shown in Figure 18.2 and the update formulas given in Equations 18.1 to 18.3.

18.3.1 Creating Meta-agents

In our system, an agent maintains a list of all its neighbours and it logs their respective interactions in so-called interaction histories. It weighs the relationships to its neighbours based on its *correlation coefficient*. A correlation coefficient between two statistical variables indicates their linear dependency. A correlation coefficient of zero implies that two variables are independent, whereas ± 1 indicates highly correlated variables. The greater the correlation between two variables, the more similar is their function. Given a series of n measurements of agents s and t in the form of s_i and t_i, where $i = 1, 2, ..., n$, their correlation coefficient (ρ_{st}) is defined as follows:

$$\rho_{st} = \frac{\sum_{i=1}^{n}(s_i - \bar{s})(t_i - \bar{t})}{(n-1)\sigma_s\sigma_t} \tag{18.4}$$

where $\bar{s}$ and $\bar{t}$ are the mean values, and σ_s and σ_t are standard deviations of s and t, respectively.

Algorithm 2 Meta-agent Creation

```
m = current_agent;
Agent new_agent;
Queue q;
q.Enqueue(m);
new_agent.Add(m);
while !q.empty() do
  Agent head = q.Dequeue();
  for all Agent s in head do
    for all Agent t in s.Neighbours() do
      if |ρ_st| ≥ τ_edge then
        new_agent.Add(t);
        q.Enqueue(t);
      end if
    end for
  end for
end while
return new_agent;
```

Algorithm 3 Validity Monitoring

```
m = current_agent;
needToBreak = false;

for all Agent s in m do
  for all Agent t in s.Neighbours() do
    if |ρ_st − ρ'_st| > τ_valid then
      needToBreak = true;
      break;
    end if
  end for
end for

if needToBreak then
  simulation.remove(m);
  for all Agent s in m do
    simulation.add(s);
  end for
end if
```

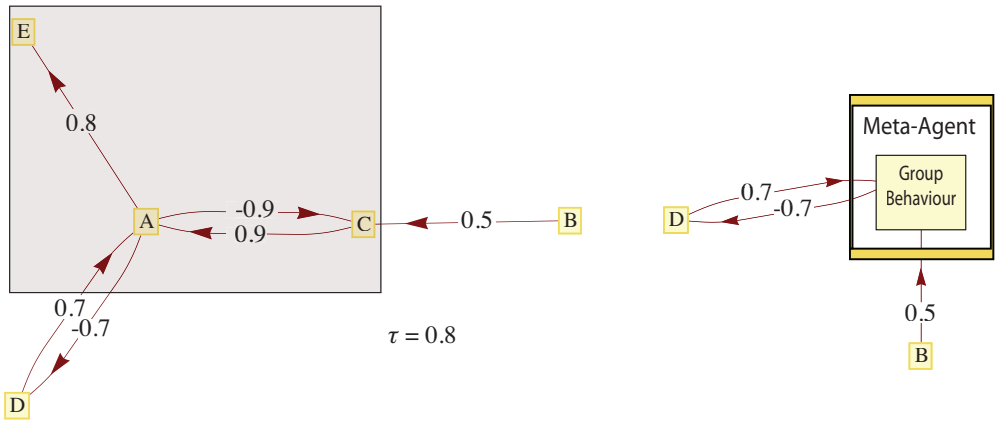
Each agent periodically checks whether its correlation coefficient with each neighbour is greater than some threshold τ_{edge}. If this is the case, they form an initial meta-agent. This heuristic process is repeated in order to identify a cluster of agents that are highly correlated (Algorithm 2). Fig. 18.3 shows an example in which *Agent A* finds *Agent C* and *Agent E*, and they form a meta-agent. The set of new neighbours is the union of all neighbours of the underlying nodes.

18.3.2 Learning the Group Behaviour

A new meta-agent replaces its underlying agents and interacts on their behalf. In order to approximate the subsumed agents' group behaviour, a learning algorithm such as artificial neural networks, evolutionary algorithms, or motif search in time series can be deployed. The learning algorithm extracts the group behaviour from the interaction histories that are locally stored with each agent.

18.3.3 Monitoring the Validity of Modules

Due to changes in the overall system, meta-agents might exhibit invalid behaviours at some point. Therefore, we check the validity of each meta-agent periodically by comparing its de-



(a) (b)

Figure 18.3: Example of an interaction graph. The edges denote the correlation coefficients. (a) *Agent A*, *Agent C*, and *Agent E* form a meta-agent, (b) The new neighbours of this meta-agent are *Agent B* and *Agent D*.

ployed behaviour with its expected behaviour. The correlation coefficients of the underlying agents serve as a heuristic indicator as they triggered the formation of the meta-agent (ρ'_{st}). According to Algorithm 3, we compare the current correlation coefficients of the meta-agent to previous values for each individual agent—if the difference is larger than some threshold, we consider the meta-agent invalid. As a consequence, we break down its hierarchy and set its underlying agents free.

18.3.4 Results

To validate the performance of our approach, we conducted a series of experiments on both MAPK models. The experiments are determined by the following five parameters (Table 18.1): We let the system run for some time t_{wait} and then start looking for meta-agents within a given time interval Δ_{find}. The waiting time t_{wait} is important as the system has to reach a rather stable condition before the abstraction algorithm starts to work. We keep monitoring the system in predefined intervals, $\Delta_{monitor}$. In order to integrate agents and to form meta-agents, the correlation coefficient between two agents—or the value of an edge in the interaction graph—should be greater than some threshold τ_{edge}. Finally, a meta-agent is valid as long as its correlation coefficients with its neighbours do not exceed the original correlation coefficients by a threshold τ_{valid}. Working values for τ_{valid} and τ_{edge} have been found through trial and error (Table 18.1).

Table 18.1: Model Parameters

Parameter Name	Symbol	Value in the 1st MAPK Model	Value in the 2nd MAPK Model
Delay before finding meta-agents	t_{wait}	1200	1500
Meta-agent finding interval	Δ_{find}	300	300
Monitoring interval	$\Delta_{monitor}$	20	20
Validity Threshold	τ_{valid}	0.1	0.1
Edge Threshold	τ_{edge}	0.95	0.7

ANN Learning

First, we present an experiment that utilizes feed-forward artificial neural networks (ANNs) with the back-propagation learning algorithm [624] to train meta-agents. The structure of an ANN is determined by its inputs and outputs as well as the number of nodes in the hidden layer. Since agents in our model are not aware of their dependent agents (they only know about their outgoing edges in the interaction graph), the output of the network should simply be all of the underlying agents. In the example shown in Fig. 18.3, outputs are *Agents A*, *C*, and *E*. The input nodes of the network are comprised of all the internal and their externally connected nodes (*Agents A*, *C*, *D*, and *E* in Fig. 18.3). As for the number of nodes in the hidden layer, we follow a simple rule-of-thumb and set it to the number of $inputs + 2$.

Fig. 18.4(a) shows the result of applying our approach to the first MAPK model in terms of the number of agents. Initially, there are eight model agents in the system. We use the term "model agent" to emphasize their role in the original model, as opposed to meta-agents that are introduced as part of the abstraction process. The identification of meta-agents starts at $t_{wait} = 1200$. The resulting pattern of periodic creation and destruction of meta-agents (Fig. 18.4(a)) stems from the fact that a meta-agent's probability to become invalid increases with its number of subsumed model agents. Generally, a meta-agent becomes invalid even if one of its subsumed agents becomes invalid. Therefore, after the system is reduced to a single meta-agent, it breaks and releases all the eight model agents.

Fig. 18.4(b) shows that the concentration computed by the agent-based pathway model successfully resembles that of the PDE solver. Fig. 18.4(c) shows the result of the same algorithm for the second MAPK pathway. Since this model is periodic, the algorithm successively finds,

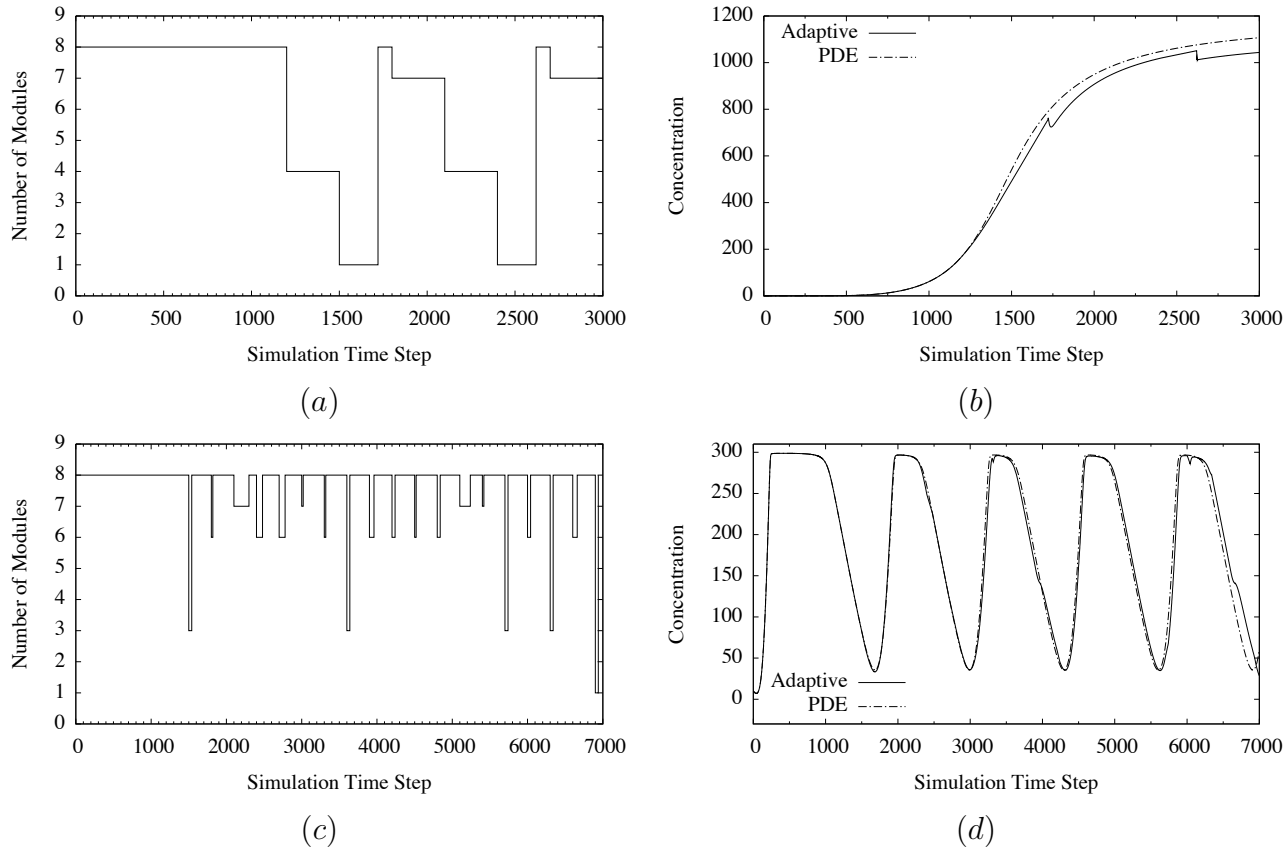


Figure 18.4: Adaptive modularization results for the MAPK pathway models of Fig. 18.1. (a), (c) Number of agents, (b), (d) Concentration of MAPK-PP.

trains, and breaks meta-agents over time. The great number of spikes in Fig. 18.4(c) implies that the meta-agents are only valid for a short period of time.

GP Learning and Dynamic Hierarchies

In a second experiment, we utilize genetic programming (GP) to find the function that approximates the group behaviour subsumed by a meta-agent. We include four mathematical operations (+, -, *, /) in the function set of the GP algorithm, whereas the internal nodes of the interaction graph serve as the available terminals. Using a heuristic learning algorithm like GP enables us to control the speed of learning and to perform a distributed search for good solutions.

The qualitative difference of this second approach compared to the presented ANN-approach is the introduction of dynamic agent hierarchies. Previously, the destruction of a meta-agent set free all the associated model agents. Now, meta-agents store references to their underlying

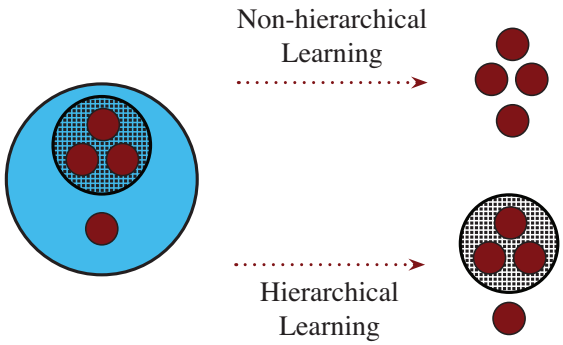


Figure 18.5: Difference of the non-hierarchical and the hierarchical approach to agent abstraction: When a non-hierarchical agent is destroyed, all the associated model agents are released back into the simulation (shown at the top). In the other case (at the bottom), the hierarchical configuration stored with a meta-agent is restored resulting in one meta-agent and one model agent.

model agents only in the first instance of the learning process. Meta-agents that subsume lower-level meta-agents store those instead, which results in a hierarchy of meta-agents with the original model agents as its leaves. When a meta-agent becomes invalid and is destroyed, its underlying agents—whether meta-agents or model agents—are released back into the simulation (Fig. 18.5).

Fig. 18.6(a) compares the performance of the hierarchical and the previously presented non-hierarchical approach. After $t_{wait} = 1200$, the non-hierarchical approach reduces the number of agents faster, but it cannot maintain any of the abstractions once the meta-agent becomes invalid at around $t = 1700$ and $t = 2500$. When the hierarchical meta-agent becomes invalid, its underlying hierarchy is restored—a single meta-agent breaks down at $t = 2200$ and releases 4 meta-agents back into the system (compared to 8 model agents). In both experiments, a meta-agent subsuming the behaviour of a larger number of agents becomes invalid very fast. This explains why an all-encompassing meta-agent does not stay long in the system ($2100 < t < 2200$ in Fig. 18.6(a)). Fig. 18.6(b) shows the MAPK concentration over time produced by the hierarchical approach and compared to the results of the PDE solver.

As Fig. 18.6(c) shows, both experiments performed similarly on the second MAPK pathway. The number of spikes in both approaches suggests that neither learning method makes a significant difference in case of periodicity. We reason that the short period of validity in both presented approaches is the result of using the correlation coefficient to measure how closely two agents work together. Since the correlation coefficient varies from -1 to $+1$ over a periodic

signal, it fails to capture the similarity of two agents in a periodic system. This result suggests that we have to look for other indicators when dealing with a periodic system.

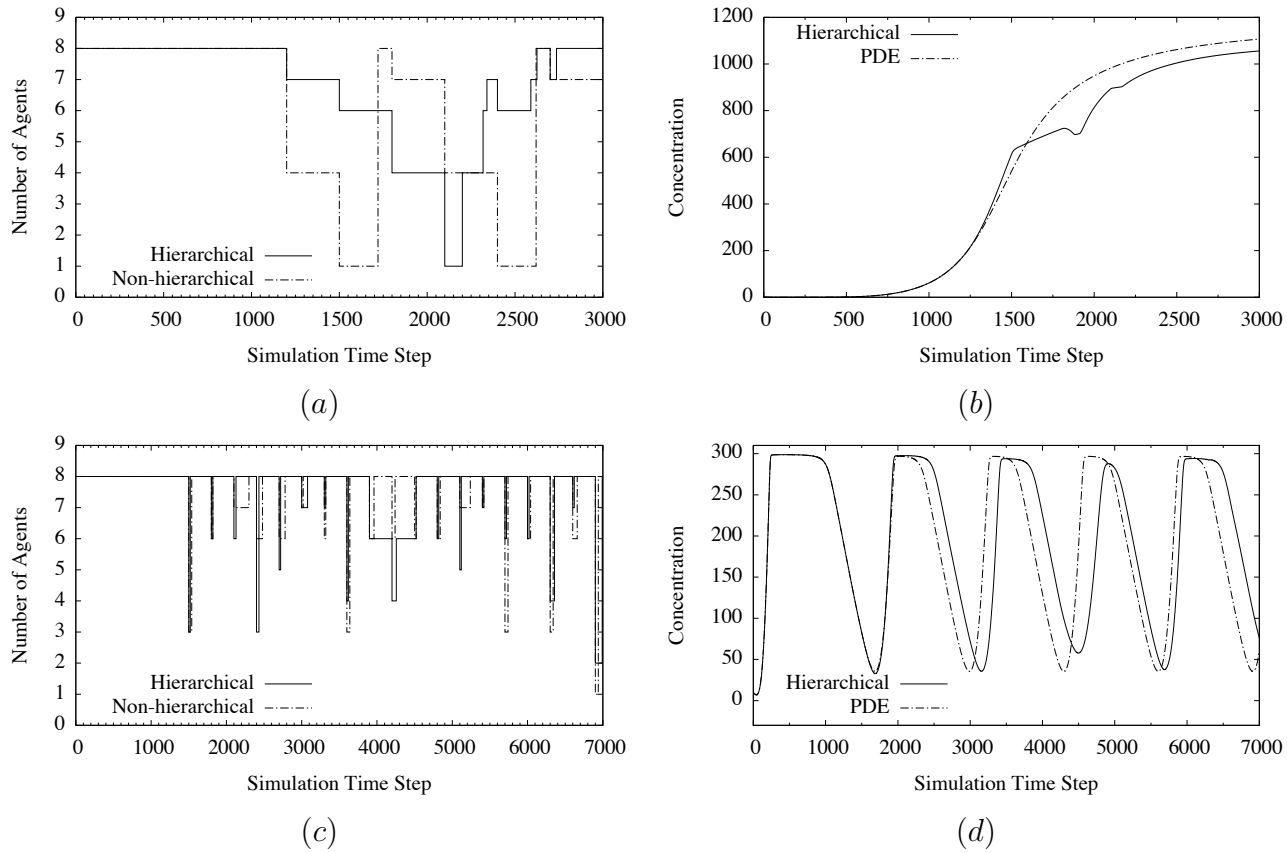


Figure 18.6: Results for the MAPK pathway model of Fig. 18.1. (a), (c) Number of agents (solid line: our hierarchical approach, dashed line: non-hierarchical approach proposed in [9]), (b), (d) Concentration of MAPK-PP.

18.4 Self-Organized Middle-Out Learning and Abstraction

In the third approach, we introduce *observer agents*, or *observers*, that coexist alongside of the model agents in the simulation space (Figure 18.7). The simulation framework treats both kinds of agents equally, i.e. each of these agent types is considered for interactions at each simulation step. Instead of an external algorithm (Section 18.3.4), the *observer agents* now handle the creation and management of abstraction hierarchies based on the interaction processes performed by model agents.

Once an *observer* successfully identifies an interaction pattern, it acts as a meta-agent that replaces the individual behaviours previously maintained by the model agents that led to the identified pattern. Acting as a meta-agent, the *observer* itself becomes subjected to observation. In order to verify their validity, *observers* would check whether the deployment of the subsumed individual behaviours would yield an outcome different from the predictions of the learned pattern. If the discrepancy between these two outcomes exceeds a given threshold, the *observer* omits its learned pattern and restores the subsumed individual behaviours.

The success of the abstraction system depends on the configuration of the deployed *observer agents*. In the following paragraphs, we explain one way how they can replace the individual behaviours with a group behaviour and how the *observer agents* can validate, maintain, or abandon the learned patterns throughout the course of a simulation.

18.4.1 Observer Configuration

Like any other agent in a multi-agent system, an *observer* can be defined as $ag = (Sit, Act, Dat, f_{ag})$, a 4-tuple composed of a set Sit of situations, a set Act of actions, a set Dat of internal data, and a decision function f_{ag} [54]. At any point in time, the agent decides to perform an action based on its situation and internal data. This decision is captured in a decision function $f_{ag} : Sit \times Dat \rightarrow Act$. In rule-based agent architectures, Dat can be re-written as Intvar $\times$ RS, where Intvar is a set of values for internal variables and RS is a set of interaction rules:

if condition **then** execute *act*,

where $act \in Act$, and condition is a statement about the situation the agent is in and the actual values of the variables in Intvar. Both condition and *act* might involve other agents called interaction partners.

Observers are configured to log the interactions of model agents in their interaction histories: IH_{Exec} is used to log executed interactions, whereas IH_{NExec} logs the numbers of considered but not executed actions (Table 18.2). An IH_{Exec} entry may contain any information related to an observed interaction. For instance, an *observer* may store that the model agent A executed an action $act \in Act$ with time stamp t along with the set of interaction partners $\mathcal{A}$.

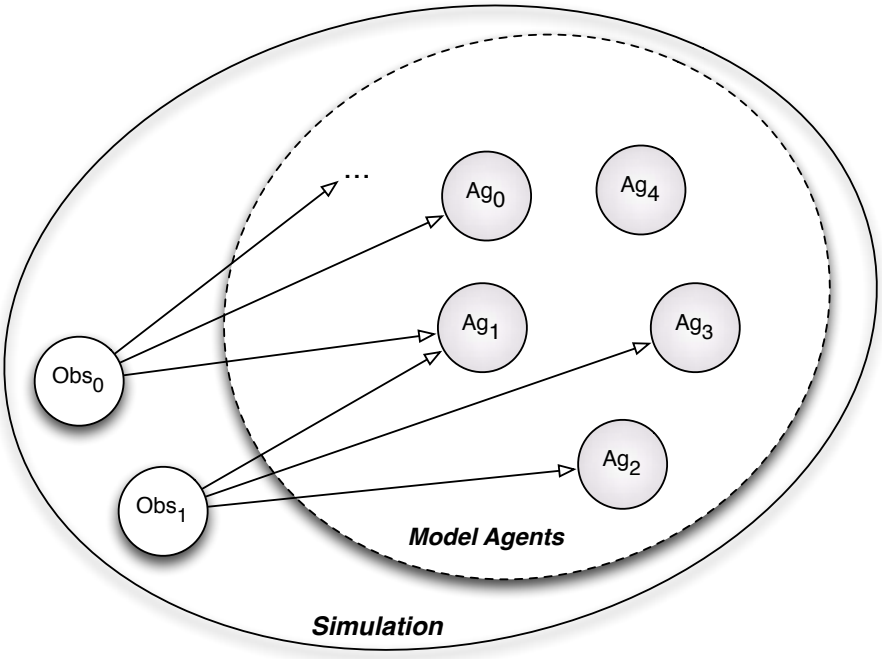


Figure 18.7: *Observers* Obs_0 and Obs_1 inside the simulation space monitor a subset of agents and log necessary information based on their configuration.

An *observer* extracts group behaviours from the logged data by applying a pattern recognition algorithm. In this section, we present results based on clustering, which will be explained next.

18.4.2 Learning and Abstraction

In our prototype, an *observer* logs interaction partners in combination with the time of the interaction. Once the interaction history IH_{Exec} has grown beyond a certain threshold, the

Table 18.2: Interaction histories inside an *observer*

Interaction History of Executed Actions (IH_{Exec})			
Time	**Agent**	**Action**	**Interaction Partners**
t_0	ag_0	Activate	$\mathcal{A}_2$
t_1	ag_0	Activate	$\mathcal{A}_2$
.	.	.	.
.	.	.	.
.	.	.	.
t_1	ag_2	Activate	$\mathcal{A}_2$
t_2	ag_7	Activate	$\mathcal{A}_2$

Interaction History of Computed but Unexecuted Actions (IH_{NExec})		
Time	**Action**	**Count**
t_0	Activate	n_0
t_3	Activate	n_3
.	.	.
.	.	.
.	.	.
t_{15}	Activate	n_{15}
t_{23}	Activate	n_{23}
t_{32}	Activate	n_{32}

observer applies a k-means clustering algorithm [602] to find a large cluster C of overlapping interaction partners. The similarity between two interactions is calculated based on the number of overlapping interaction partners. When the *observer* finds such a cluster, it infers a generalized group behaviour from the clustered individual interactions by combining their features. The first feature is the set of overlapping interaction partners that are constant for the learned action. Secondly, the *observer* needs to know when and at which rate it should execute the learned action.

The *observer* first finds the time range $[t_{min}, t_{max}]$ of the executed action from all the individual interactions in C. Two cases might happen here: (1) an interaction only occurs within a bound time range, (2) an interaction continuously occurs over time or the *observer* is uncertain whether it has had enough time to determine an upper bound t_{max} of the time range. In order to address the latter case, the *observer* compares the two most recent time stamps an interaction occurs in $\mathrm{IH}_{\mathrm{Exec}}$. If the difference exceeds the observation time, the *observer* sets t_{max} to ∞.

Next, the *observer* extracts the rate of execution defined as the number of executed interactions divided by the number of total computations of the interaction:

$$p_{exec} = \frac{|\mathrm{IH}_{\mathrm{Exec}}|}{|\mathrm{IH}_{\mathrm{Exec}}| + |\mathrm{ihn}|},\ \mathrm{ihn} \in \mathrm{IH}_{\mathrm{NExec}}\ \&\ \mathrm{ihn}.t \in [t_{min}, t_{max}] \tag{18.5}$$

where $\mathrm{IH}_{\mathrm{Exec}}$ is the set of executed interactions, ihn is the set of considered but not executed interactions whose timestamp is in $[t_{min}, t_{max}]$, and $|\cdot|$ denotes the size of a set.

For example, all the $\mathrm{IH}_{\mathrm{Exec}}$ records in Table 18.2 constitute a cluster in which $[t_{min}, t_{max}]$ is inferred from the first column. The second column (ags) is discarded and regarded as wildcard, the set of interaction partners is fixed to $\mathcal{A}_2$, and p_{exec} is calculated as described above.

Finally, the *observer* removes the action *act* from the model agent. From now on, it performs the action on the model agent's behalf. For instance, an *observer* may learn that action *act* of an agent A occurs at $A.t \in [t_{min}, t_{max}]$, and executes it on A's behalf with an according probability p_{exec}. Since the *observer* also learns the interaction partners an action depends on, the computational resources to identify those are saved as well.

18.4.3 Validation of the Learned Behaviours

After some time, a learned behaviour might not be valid any more. In order to monitor the reliability of a learned behaviour, it is initialized with an unbiased confidence value $conf_{initial} = 50\%$. At regular time intervals, the *observer* lets some model agents execute their original interactions. The confidence value is regulated based on the difference between the actual behaviour of model agents compared to the behaviour expected by the *observer* (Fig. 18.8). In our prototype, we only consider the difference in interaction partners. The time at which an individual interaction occurs or the rate at which model agents execute their interactions could also be incorporated into the comparison. A confidence measure below a given threshold indicates that a learned group behaviour is not valid any longer and that the *observer* has to restore the model agents' original behaviours.

18.4.4 Experiments

The outlined self-organized optimization method can be employed in arbitrary agent simulations. Biological simulations are particularly suitable applications as biological entities will be directly modelled as agents. When simulating biological systems at the level of inter-cellular and inter-molecular interactions, actions are mostly triggered by collisions or internal agent states. We applied our proposed method to an agent-based simulation of blood coagulation described in the next subsection. All the experiments were repeated 10 times to ensure that a particular experiment did not bias the results.

Model Setup

Blood coagulates at wound sites because of the interplay of various bio-agents, e.g., platelets, fibrinogens, and serotonins. If a collagen protein collides with a platelet, the platelet becomes activated. In case an activated platelet collides with the wound site, it secretes several chemicals which in turn activate more platelets in the blood vessel. Gradually, a network of fibrins together with a platelet plug form a clot around the wound site (Fig. 18.9). We modelled twelve blood factors as agents whose behaviours are expressed as a set of interaction rules. There are ten different interactions which fall into two categories: (1) state-dependent interactions and (2)

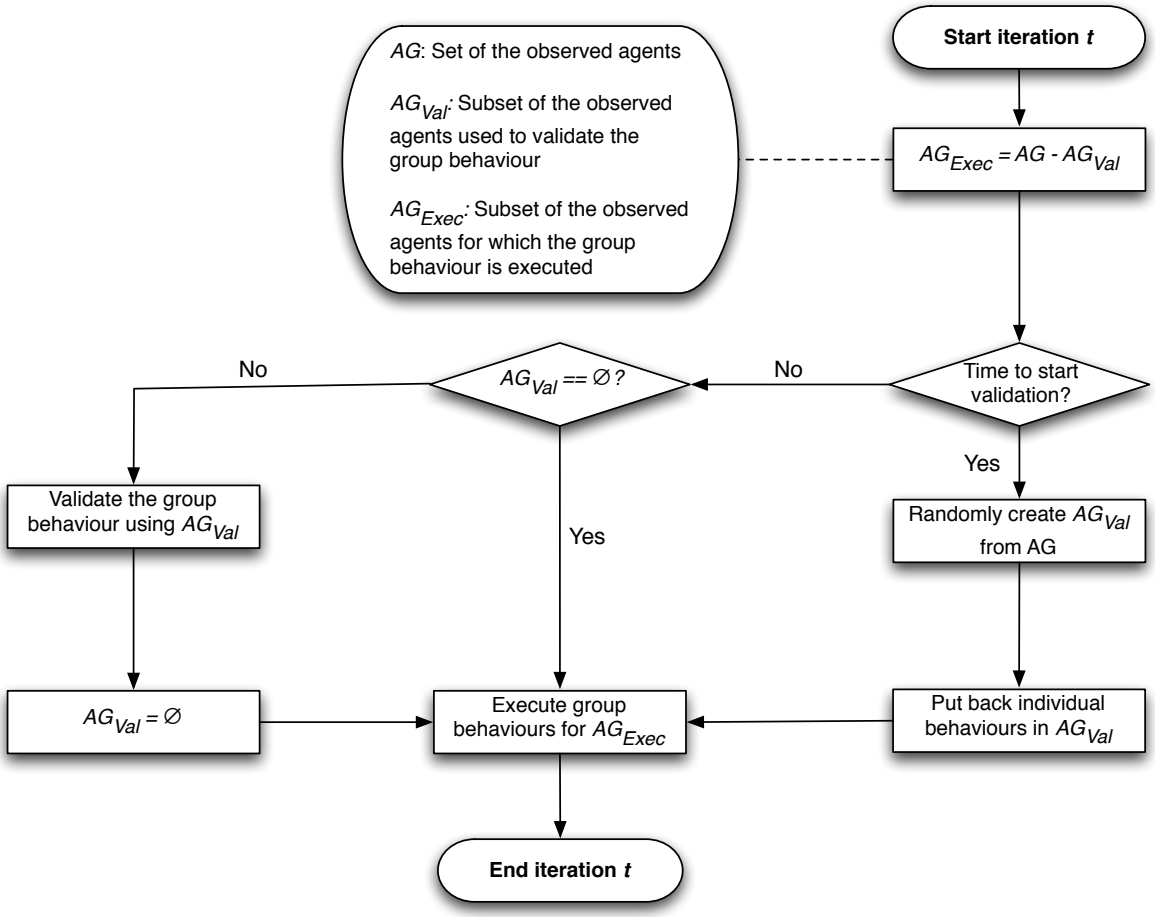


Figure 18.8: Flow chart of the validation step. At some interval, the *observer* selects a random subset of the observed agents and restores their individual behaviours. The result of their interactions is evaluated in the next iteration to regulate the confidence value. The *observer* continues to execute the group behaviour for all other agents.

collision-dependent interactions. The actions themselves introduce local state changes of the agents (represented as internal variables), or they produce or remove agents in the simulation. The simulation starts with 10 agents and ends up with nearly 140 interacting agents (Fig. 18.10).

Observer Setup

Each interaction is monitored by an *observer* that records only the interaction partners. Table 18.3 lists all the important parameters in our system. Once an *observer* monitors an interaction long enough (t_{wait}), it applies a k-means clustering algorithm to create k clusters. The centroid of the largest cluster is considered to be the learned group behaviour for which $[t_{min}, t_{max}]$ and

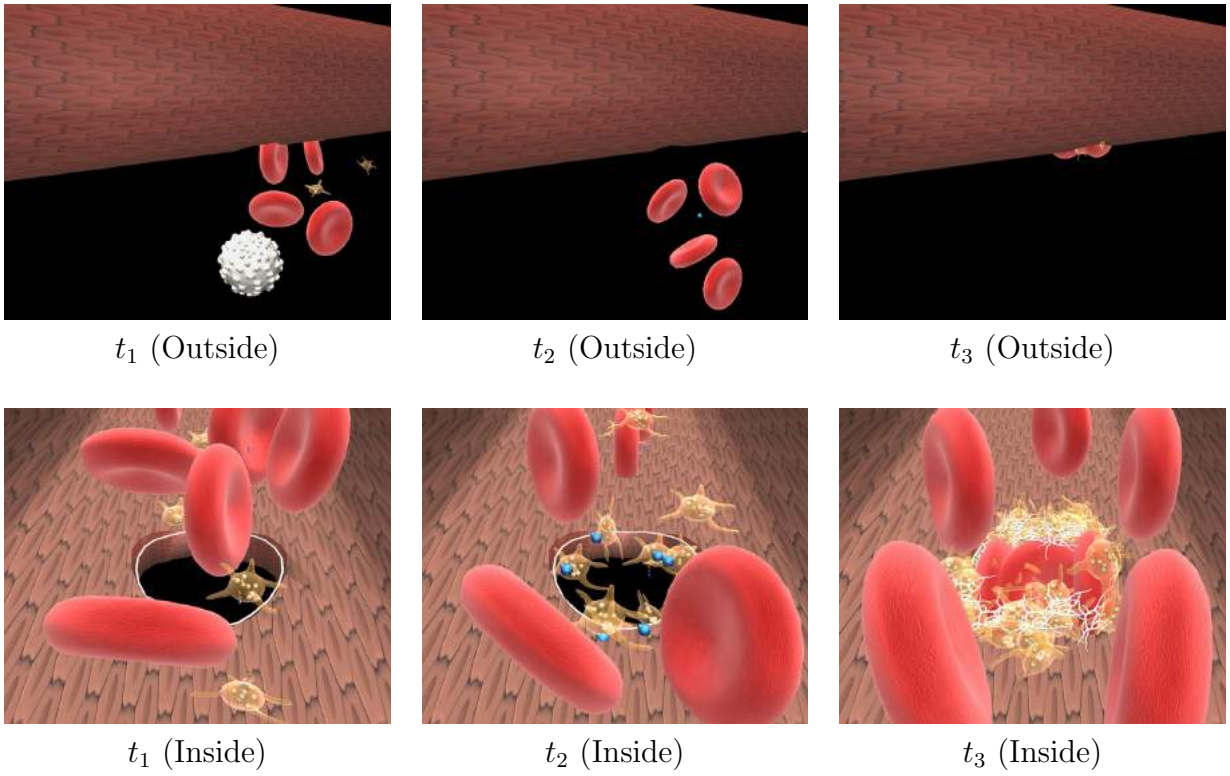


Figure 18.9: The blood coagulation simulation at different time steps ($t_1 < t_2 < t_3$). The process is observed from two different perspectives: inside and outside of the vessel.

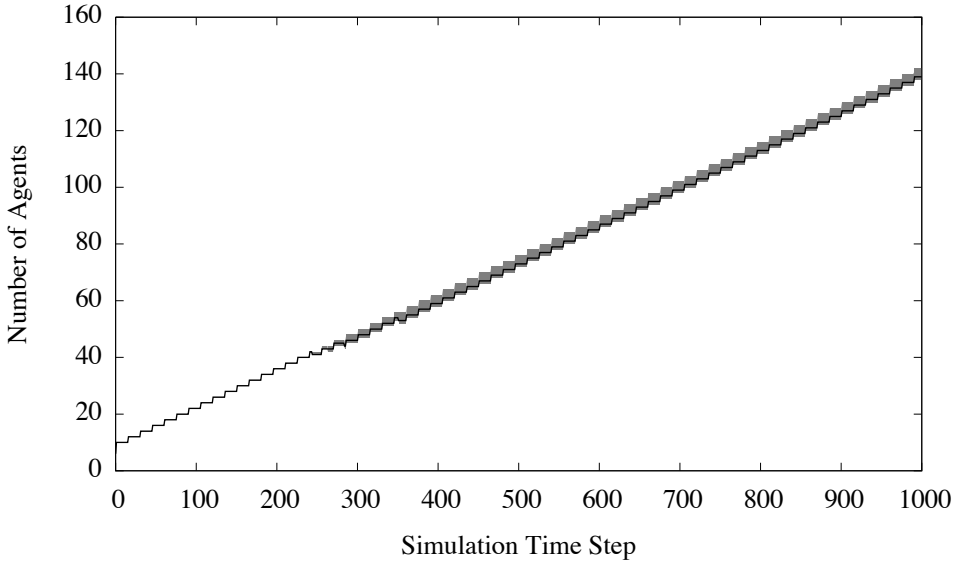


Figure 18.10: Blood coagulation simulation: Number of agents over time.

Table 18.3: System Parameters

Parameter Name	Symbol	Value
Delay before learning	t_{wait}	350
Validation interval	$V_{interval}$	70
Validation length	V_{length}	10
Validation ratio	V_{ratio}	30%
Confidence threshold	τ_{conf}	0.3
Number of clusters in k-means	k	10

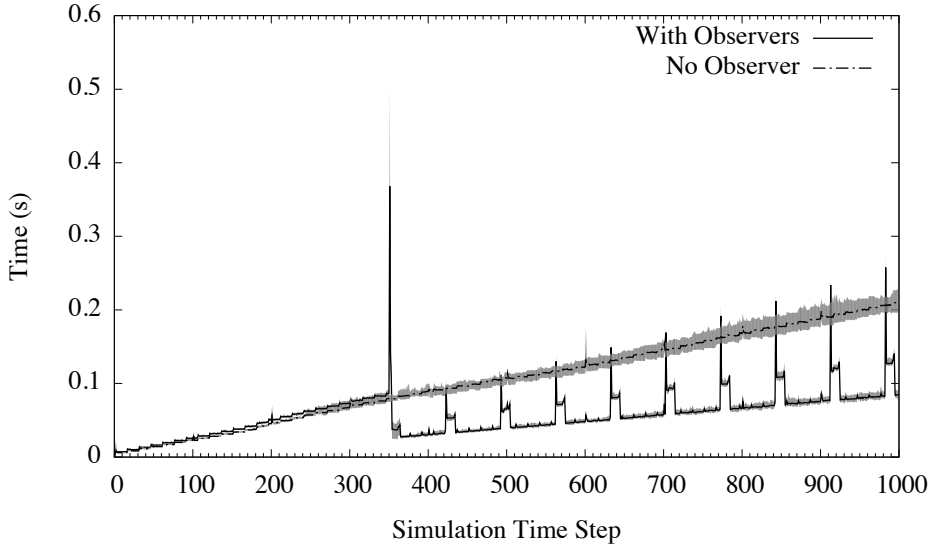
p_{exec} are inferred. The *observer* subsumes the learned interaction by executing it on behalf of the model agents. In predefined intervals, $V_{interval}$, the *observer* randomly chooses a subset of the subsumed behaviours and allows the model agents to execute their original interactions. The size of this subset is determined by V_{ratio}. After some time, V_{length}, the *observer* subsumes this subset again and and validates its abstractions based on the resulting interaction compared to the expected result. The confidence of the learned pattern is regulated accordingly. If the confidence of a pattern is less than some threshold τ_{conf}, the learned pattern will be removed from the simulation.

Results

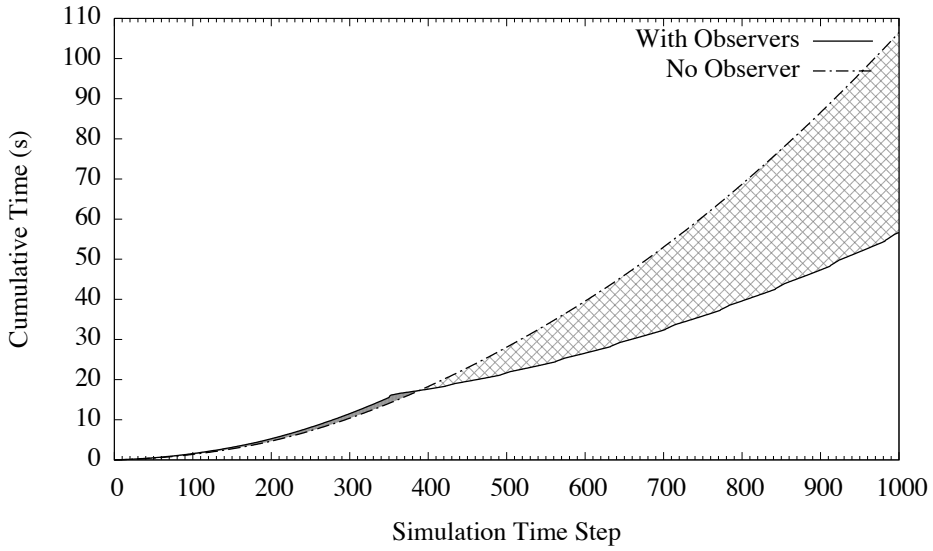
The presented prototype implementation successfully identified several group behaviours within the simulation. For example, **Random Walk** is a self-triggering action found to be executed with probability $p_{exec} = 100\%$ and $t \in [0, \infty]$. **Adhere** is an interaction executed in $t \in [172, \infty]$ with probability $p_{exec} = 65\%$. **Self Activation** is another collision-based example with $p_{exec} = 2.8\%$ and $t \in [173, 190]$.

Figure 18.11(a) shows the actual run-time of the simulation at each time step. When there is no *observer*, the simulation slows down as it proceeds, as simulating the interactions among the increasing number of agents requires more computations. When the *observers* are present in the simulation logging interaction data ($0 < t < 350$), they add a little overhead to the run-time of the whole simulation. At $t = 350$ when the learning happens, there is a peak in the run-time. However, once successfully deployed, the *observers* reduce the run-time drastically by executing group behaviours instead of individual behaviours. The validation cycle is triggered every $V_{interval} = 70$ time steps, therefore there is a fairly high peak at this interval. It continues for

$V_{length} = 10$ time steps before the learned pattern is evaluated. After this time, the simulation runs fast again until the next validation cycle.



(a)



(b)

Figure 18.11: Blood coagulation simulation: Run-time with and without *observers*, (a) Run-time per simulation time step, (b) Cumulative run-time.

Figure 18.11(b) depicts the cumulative run-time of the simulation comparing a normal run against a run with *observers*. The overhead of having *observers* clearly pays off at $t > 390$, when the cumulative run-time of a normal run exceeds that of a run with *observers*. On average, a normal run takes 107 seconds to complete 1000 simulation time steps, almost twice as long as the run with the *observers*, which takes 56 seconds.

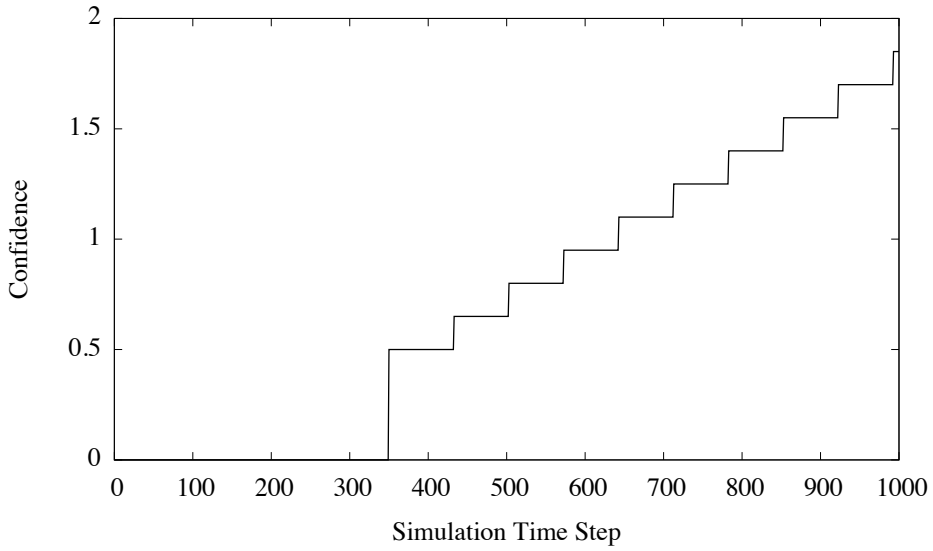


Figure 18.12: The confidence value over time shown for an exemplarily learned pattern.

Figure 18.12 shows the change of confidence for one of the learned patterns. Since there is no learned pattern before $t = 350$, the confidence value is also 0. However, after the *observer* abstracts an individual behaviour, the confidence value is initially set to 0.5. As all the abstractions work correctly, the confidence values continuously increase over time.

18.5 Conclusion and Future Work

We introduced a concept for the reduction of computational complexity in agent-based models by means of learning behavioural patterns over the course of a simulation. The abstractions would be expressed as meta-agents that subsume lower-level agents and be seamlessly integrated into the agent models.

We presented and evaluated three implementations: (1) The first utilized artificial neural networks to learn collective processes in the flux of concentrations of the MAPK signaling pathway. Here, the learned abstractions were constantly updated to consider a growing number of agents. As a result, the abstractions lost their validity at some point, they were removed from the simulation and relearned. (2) In the second implementation, which relied on genetic programming for learning collective behaviours, the abstractions were not completely revoked when becoming invalid, but they were restored to their previous states. (3) In the third implementation, *observer* agents detected group behaviours and managed the resulting abstractions. We demon-

strated the effectiveness of this implementation in the context of a blood coagulation model. We proposed two algorithms to monitor the validity of abstractions by comparing the expected group interactions to the interactions of the actual individuals at regular intervals.

In order to further our approach, we suggest the automatic proliferation of a diverse set of *observer agents* based on their workload. An evolution of agents that are primed to identify frequently occurring patterns could be implemented, yielding a self-organized learning system that adapts to specific model domains and even to niches inside of simulation spaces.

The relation between group behaviours and emergent phenomena is another promising area to be investigated in the given context. The possibility to incorporate predefined high-level patterns should be considered. If patterns are described at different scales, multi-scale modeling can be restated as finding transitions from low-level to higher-level patterns.

Chapter 19

Adaptive Agent Abstractions to Speed Up Spatial Agent-Based Simulations

Simulating fine-grained agent-based models requires extensive computational resources. In this article, we present an approach that reduces the number of agents by adaptively abstracting groups of spatial agents into meta-agents that subsume individual behaviours and physical forms. Particularly, groups of agents that have been clustering together for a sufficiently long period of time are detected by *observer* agents and then abstracted into a single meta-agent. *Observers* periodically test meta-agents to ensure their validity, as the dynamics of the simulation may change to a point where the individual agents do not form a cluster any more. An invalid meta-agent is removed from the simulation and subsequently, its subsumed individual agents will be put back in the simulation. The same mechanism can be applied on meta-agents thus creating adaptive abstraction hierarchies during the course of a simulation. Experimental results on the simulation of the blood coagulation process show that the proposed abstraction mechanism results in the same system behaviour while speeding up the simulation.

Abbas Sarraf Shirazi, Timothy Davison, **Sebastian von Mammen**, Jörg Denzinger, and Christian Jacob. *Adaptive agent abstractions to speed up spatial agent-based simulations.* Simulation Modelling Practice and Theory 40 (2014), pp. 144–160.

19.1 Introduction

Agent Based Models (ABM) provide a natural means to describe complex systems, as agents and their properties have a convenient mapping from the entities in real world systems. The interaction of agents in ABM gives rise to an interesting concept in the study of complex systems: emergent phenomena, higher-level properties or behaviours that are not easily traceable in the lower-level entities [89]. Moreover, ABM capture discontinuity in individual behaviours, which is difficult when modelling with an alternative like differential equations [241].

The flexibility of ABM comes at a computational cost. As the granularity of a model increases, so do the computational resources needed to simulate all of the interactions among the agents, which directly translates into longer simulation times. Some researchers have restricted agent interactions to be only among neighbouring agents in a two or three-dimensional lattice [625, 607]. However, changing the interaction topography among agents is a necessary feature in some models, e.g. developmental processes [342]. Others have utilized parallel computing to meet the computational demands of ABM [626, 627]. Finally, many researchers have proposed super-individuals [259]: agents that encompass other agents, e.g. a single super red blood cell agent that subsumes and represents thousands of individual red blood cell agents.

In this paper, we extend our previous work by proposing another type of abstraction that aims to build adaptive hierarchies of spatial agents during the course of the simulations. To this end, *observer* agents are immersed in the simulation to monitor groups of agents. The *observers* try to detect a cluster of agents that have adhered to one another for a sufficiently long duration of time. Once an *observer* finds such a cluster, it abstracts the agents into a single meta-agent that subsumes both the behaviour and the structure of the individual agents in that cluster. As the dynamics of the simulation change, groups of agents may no longer stick together and therefore the *observer* needs to break down those meta-agents into their constituent individual agents. An unsupervised validation mechanism ensures the validity of meta-agents by periodically monitoring whether they should continue to subsume their agents. Since meta-agents have the same basic definition as the individual agents, the same abstraction process is applied on them, thus making adaptive abstraction hierarchies during the course of the simulation.

The remainder of this paper is organized as follows. Section 19.2 reviews related works both in solving the problem of scalability and in dealing with higher-order patterns in agent-based simulations. Section 19.3 gives a formal definition, along with a computational timing analysis of our component-based agent framework – *LINDSAY Composer*. Section 19.4 presents our abstraction framework with a detailed description of the involved steps and algorithms. We conclude this section with a computational timing analysis of our abstraction. In order to demonstrate the effectiveness of this approach, we apply it to an agent-based blood coagulation simulation and report the results in Section 19.5. Finally, section 19.6 provides a comparison between this work and our previous work, and presents the concluding remarks.

19.2 Related Work

Agent based models operate at the individual level and describe potentially numerous behaviours for all of their constituent units. Simulating all of the individual behaviours is therefore considered to be extremely computationally intensive [241, 242, 243, 244, 628]. It has been suggested that abstracting higher-order patterns could reduce the computational complexity of ABM without introducing much overhead [247, 628, 516]. In this section, we briefly describe the attempts made to address the problem of scalability and performance in ABM, then we review the works that motivated this research.

19.2.1 Scalability and Performance in ABM

Bonabeau points out that despite increasing computational power, simulating all the individual behaviours in ABM still remains a problem when it comes to modelling large-scale systems [241]. Research in improving the scalability of ABM is roughly categorized into two groups: (1) parallel computing, and (2) grouping similar agents into a single agent.

The first category, parallel computing, tries to concurrently simulate clusters of agents that interact primarily with one another without much intra-cluster communication. Efficiency is improved as long as the time spent on synchronization is much less than the time spent on computation [626]. Scheutz and Schermerhorn developed a framework with two algorithms for the automatic parallelization of ABM [626]. Particularly, they developed a separate algorithm

for spatial agents, as their location data can efficiently determine in what cluster they should be simulated.

Along the same line, Lysenko and D'Souza propose a framework to use Graphics Processing Units (GPU) to parallelize an agent-based simulation [627]. They utilize a technique in General Purpose Computing on GPUs called state textures [629] to map each agent to a pixel. A pixel is defined by its colour components: Red, Green, Blue, and Alpha (RGBA). Each numerical property of an agent is thus mapped to a colour component. If an agent cannot be squeezed into four floating point values, then extra colour buffers should be used, which in turn adds to the complexity of the problem.

The second category of grouping similar agents deals with the granularity of an agent. For example, super-individuals can represent groups of agents. Scheffer et al. suggest assigning an extra variable to each agent to denote how many agents it represents [259]. More advanced algorithms have been proposed to find super-individuals during the course of a simulation. Stage et al. propose an algorithm called COMPRESS to aggregate a cluster of agents into one agent [513]. They divide their algorithm into two stages to avoid applying a time-consuming clustering algorithm on the space of all the attributes in all the agents. In the first stage, they calculate a linear combination of attributes l_i for each agent i by applying principal component analysis (PCA) [260]. Then this list is sorted to find n clusters of agents with the largest gaps in l_i. In the next stage the clusters are further subdivided based upon their variance until the variance is within a given range. The first stage maintains overall system variations while the second stage reduces the intra-cluster variations.

COMPRESS is a static algorithm, in that once a cluster of agents is replaced by one agent, the original agents will not be released back into the simulation. Wendel and Dibble extend the static COMPRESS algorithm with the Dynamic Agent Compression (DAC) algorithm in which higher-order agents are created and destroyed based on the heterogeneity of agents in the system [514]. They define two special agents in their system: (1) *container agents* which are the higher-order agents, and (2) a *compression manager* which handles all the queries to individual agents thus making the container agents invisible to the model. It also creates and destroys other agents. For example, upon receiving a create request from the model, the compression manager decides if it has to create a new individual or whether the create request can be ignored, as there already exists an agent with the same attributes. In DAC, a container

agent monitors its encompassed agents, and upon detecting a difference in behaviour, gives the changed agents to the compression manager as newly instantiated individuals.

In a previous work [516], we have shown the speedup in the simulation by abstracting individual rules in the agents. Particularly, we replace many individual rules with one stochastic meta-rule that only depends on the simulation time step. To that end, *observers* are preconfigured to monitor certain rules in the simulation. The *observers* look for interaction patterns, i.e rules whose action is executed with constant parameters. Once an observer successfully identifies an interaction pattern, it acts as a meta-agent and replaces all the individual rules previously maintained by the agents by a new stochastic meta-rule. Although this abstraction does not create agent hierarchies, it results in a speedup in the simulation.

19.2.2 Higher-Order Patterns in ABM

Emergence is the appearance of macro level patterns in a system that are not described by the properties of its parts [278]. According to Müller [630], an emergent phenomenon is either observed by an external observer (weak emergence), or by the agents themselves (strong emergence), provided that they have the knowledge to describe it. ABM provide a basis to observe emergent phenomena, as the behaviour of the modelled system can be traced during the execution. In this sub-section, we describe a few examples of higher-order, emergent phenomena modelled with ABM.

Cellular automata are one of the most widely used tools to study macroscopic patterns that emerge from the discrete, microscopic interactions in a 2D or 3D lattice [631]. While continuous models (e.g. differential equations) fail to capture the essentials of certain problems like self-reproduction of cells, such phenomena can be studied when modelled as cellular automata [632]. Although each agent is restricted to interact with its local neighbours, numerous higher-order patterns have been studied using cellular automata, such as self-organization in Conway's Game of Life [633], pattern formation in biological systems [634], engineering applications [635], and medical simulations [636].

ABM can mimic the behaviour of mathematical models and at the same time they give more power to the modeller. Bonabeau states that there is not much experimental work in the field of pattern formation in spite of a large body of theoretical analysis [637]. He claims that ABM

can bridge this gap, as they are amenable to experimental observations. Subsequently, he shows how to derive the equivalent agent-based representation of a reaction-diffusion model. Agents in his model perform a random walk and interact with the environment by depositing or removing bricks at a rate calculated from the mathematical formula of a reaction-diffusion system. He takes the agent-based model beyond its original reaction-diffusion system by replacing the state-dependent variables with short and long term memories in the agents.

Generally, higher-order patterns emerge when spatial agents act as a group. For example, in crowd modelling, groups of people tend to walk together while keeping their distance from other groups [638, 639]. Social segregation [640] is another example in which different social (ethnical, racial, or religious) groups tend to avoid other groups. Studying these systems promotes tolerance and social integration [641]. In an example from biological systems – blood coagulation – the adhesion of platelet and fibrinogen molecules leads to the formation of a blood clot within a damaged blood vessel wall [642]. One may observe a clot as an emergent entity formed as the result of interactions among several other smaller entities [605].

19.3 Agent Formalism

In this section, we formally describe our concept of agents. A formal definition of agents helps us clarify our component-based agent architecture, e.g. how we define an agent with regards to its constituent components, the interdependency among components, etc. It further provides a basis to analyze the computational complexity of our simulations. The timing analysis is used in the next section to study the benefit of the abstraction mechanism in improving the run-time of a simulation.

We use a generic definition of agents and show how our component-based composition of an agent fits into this definition. In our framework, an agent is defined by a 4-tuple:

$$agent = (\mathbf{Sit}, \mathbf{Act}, \mathbf{Dat}, f) \tag{19.1}$$

where **Sit** is the set of situations the agent can be in, **Act** is the set of actions that it can perform, **Dat** is the set of value combinations for its internal data areas, and f is a decision function [279]. At any point in time an agent decides what actions to perform based on its

current situation and its internal data. This decision is captured by the decision function $f : \mathbf{Sit} \times \mathbf{Dat} \rightarrow \mathbf{Act}$.

We employ the component-based approach introduced in *LINDSAY Composer* [305, 540] to construct the agents in our simulations. A component, $comp^i$, is a mini-agent that can be combined with other components to create agents with aggregate functionalities:

$$comp^i = (\mathbf{Sit}^i, \mathbf{Act}^i, \mathbf{Dat}^i, f^i) \tag{19.2}$$

With this in mind, an agent is re-defined as the composition of several components:

$$agent = \langle comp^1, comp^2, \cdots \rangle \tag{19.3}$$

$$\mathbf{Sit}^{agent} \subseteq \mathbf{Sit}^1 \times \mathbf{Sit}^2 \times \cdots \tag{19.4}$$

$$\mathbf{Act}^{agent} \subseteq \mathbf{Act}^1 \times \mathbf{Act}^2 \times \cdots \tag{19.5}$$

$$\mathbf{Dat}^{agent} \subseteq \mathbf{Dat}^1 \times \mathbf{Dat}^2 \times \cdots \tag{19.6}$$

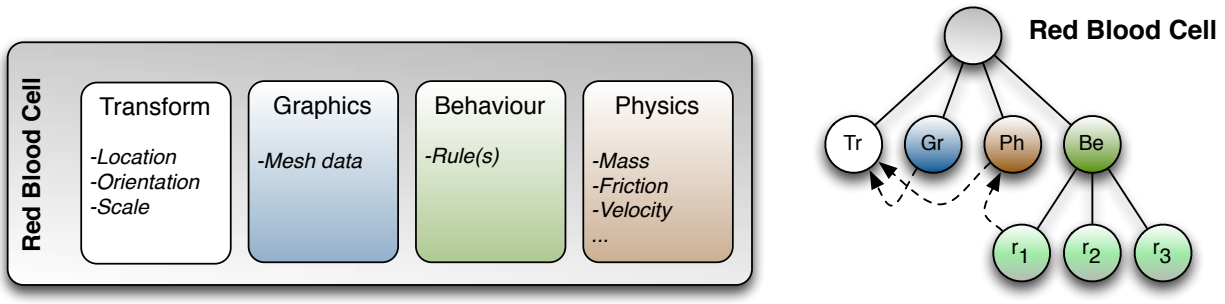
$$f^{agent} = \langle f^1, f^2, \cdots \rangle = \langle f^1 : \mathbf{Sit}^1 \times \mathbf{Dat}^1 \rightarrow act^1, f^2 : \mathbf{Sit}^2 \times \mathbf{Dat}^2 \rightarrow act^2, \cdots \rangle \tag{19.7}$$

where $\mathbf{Sit}^{agent}$ is a subset of all the combinations of $\mathbf{Sit}^i$ in the components of an agent. While $\mathbf{Act}^{agent}$ and $\mathbf{Dat}^{agent}$ are defined similar to $\mathbf{Sit}^{agent}$, $\mathbf{f}^{agent}$ is defined as a vector of all the decision functions. In other words, all the actions chosen by the individual decision functions will be executed by the agent. It should be noted that there is no internal conflict resolution between conflicting actions. It is up to the system builder to avoid composing conflicting actions.

Behaviour components use a rule-based architecture in which **Dat** is re-written as $\mathbf{Intvar} \times \mathbf{RS}$, where **Intvar** is a set of values for internal variables and **RS** is a set of interaction rules, as defined in Equation (19.8).

$$\mathbf{RS} = \{(r_1, ..., r_k) \mid r_i : \textbf{if } \texttt{condition}_i \textbf{ then } \texttt{execute } act_i\} \tag{19.8}$$

where $act_i \in \mathbf{Act}$, and condition_i is a statement about the situation the agent is in and the actual values of the variables in **Intvar**.



(a) (b)

Figure 19.1: An example of a red blood cell agent which is composed of transform, graphics, physics, and behaviour components. (a) The internal data in each component, (b) The inter-dependency of the sibling components in the hierarchy is denoted by dashed lines. Tr, Gr, Ph, and Be stand for the transform, graphics, physics, and behaviour components, respectively.

For example, the red blood cell agent in Figure 19.1(a) is defined as a combination of four sibling components. (1) A transform component containing all the information necessary to represent an agent in a three-dimensional space. (2) A graphics component with the data about how to render this agent, e.g. the mesh data. (3) A physics component containing all the physical properties like the mass and friction, which enables this agent to undergo physical interactions with other physical agents. (4) A behaviour component which includes the set of interaction rules **RS**.

A component may depend on the situation or the data of another component. The dependency of a component to other components is encoded in its **Sit** and **Dat**, i.e. there might be $\mathbf{Sit}^i$, $\mathbf{Dat}^i$, or subsequently f^i in a component that looks up areas in $\mathbf{Sit}^j$ and $\mathbf{Dat}^j$ of another component. In Figure 19.1(b), the graphics component depends on the transform component to render a mesh while the physics component updates the same transform component once a physical force is applied to this agent. The physics component might also be used to trigger the execution of a custom rule, e.g. an action by the agent when it collides with another agent.

Components in *LINDSAY Composer* may delegate their decision functions to an engine which in turn drives their execution at each frame of the simulation. Specifically, a component may decide to share any of its $\mathbf{Sit}^i$, $\mathbf{Dat}^i$, $\mathbf{Act}^i$, or f^i with an engine. The link between a component to an engine is also encoded in $\mathbf{Dat}^i$, which specifies what engine to delegate to, along with the parameters for that engine. This design explicitly formalizes the link between components and engines. It also gives the components the freedom to share an engine or to instantiate new

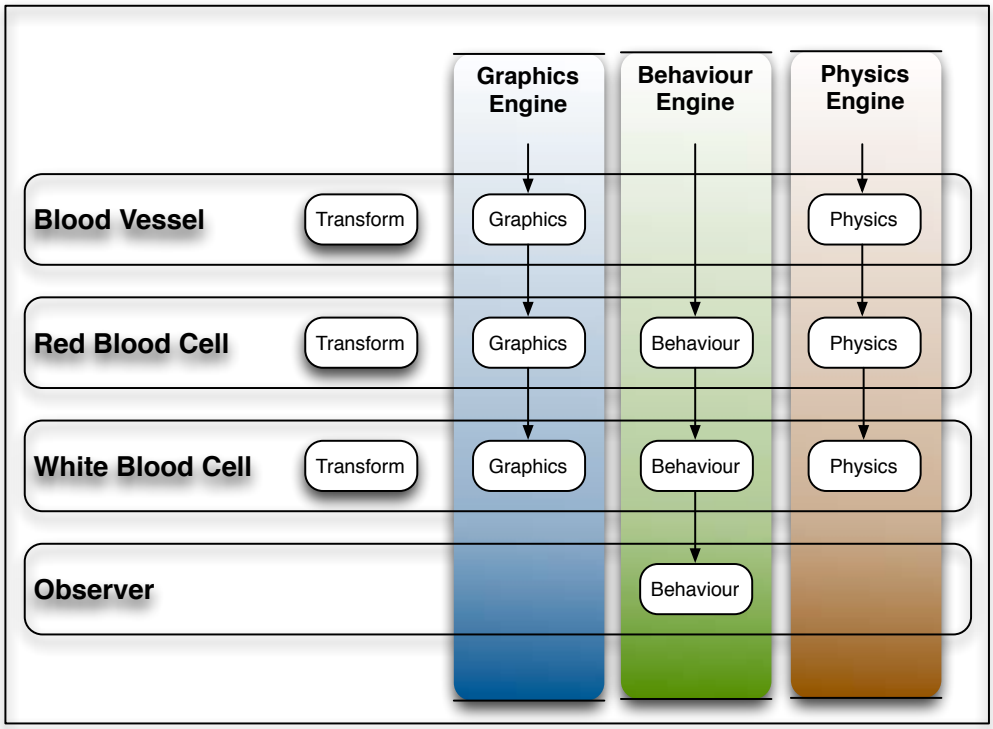


Figure 19.2: Three default engines in *LINDSAY Composer* drive the execution of components within agents.

engines based on their needs.

LINDSAY Composer includes the following default core engines which are instantiated once the simulation starts: (1) the graphics engine which renders all the graphics components onto the screen, (2) the physics engine which handles all the physics components, and (3) the behaviour engine which executes the rules in the behaviour components. In the simple scenario of a single scale simulation, each engine iterates through its components and updates them at each frame. Figure 19.2 shows how each engine drives the execution of the delegating components.

The explicit formalism of components, agents, and engines in *LINDSAY Composer* makes it easy to analyze the performance of the agent-based simulations. The time required to simulate an agent-based model in *LINDSAY Composer* with a single processor is defined as follows:

$$TotalTime = T_{init} + \sum_{t=1}^{T} Step^t \tag{19.9}$$

$$Step^t = \sum_i step^t(eng_i) \tag{19.10}$$

where the initial time, T_{init} is the time spent only at the beginning of the simulation – for example to instantiate the default engines – and T is the length of the simulation.

The graphics and physics engines are usually well-optimized, as their functionality is limited to rendering and calculating the physical forces acting upon objects. On the other hand, the behaviour engine is where every custom behaviour of an agent is executed, and is therefore the bottleneck of the simulation. As a result, we only focus on the performance of the behaviour engine:

$$Step^t = step^t(BehaviourEngine) = \sum_{i=1}^{N} BComp^{t,i} = \sum_{i=1}^{N} \sum_{j=1}^{R^i} r_j^{t,i} \tag{19.11}$$

where N is the number of behaviour components, $BComp^{t,i}$ is the i^{th} behaviour component and $r_j^{t,i}$ is the j^{th} interaction rule in $BComp^{t,i}$ at time step t.

Without loss of generality, one can assume that all behaviour components have the same number of interaction rules, i.e. $R = \max(R^i)$. The asymptotic complexity of simulating the behaviour engine at each time step is calculated as follows:

$$O(Step^t) = \sum_{i=1}^{N} O(BComp^{t,i}) = \sum_{i=1}^{N} \sum_{j=1}^{R} O(r_j^{t,i}) = \sum_{i=1}^{N} \sum_{j=1}^{R} O(1) = N * R \tag{19.12}$$

Equation (19.12) is a lower bound of $O(Step^t)$ since it assumes that the computational complexity of executing each interaction rule is $O(1)$. This assumption is correct when it takes $O(1)$ for an agent to check the condition of a rule. In other cases, agents might need to iterate over every other agent in the simulation to check whether the condition part of a rule holds or not, hence Equation (19.12) changes to $O(Step^t) = N^2 * R$. This formula clearly shows that the run-time of a simulation mainly depends on (a) the number of agents in the simulation, and (b) the number of interaction rules for each agent. It can also be inferred that simulating all the behaviours for all the agents requires a great deal of computational resources. We propose an abstraction mechanism to address this issue, which is discussed in the next section.

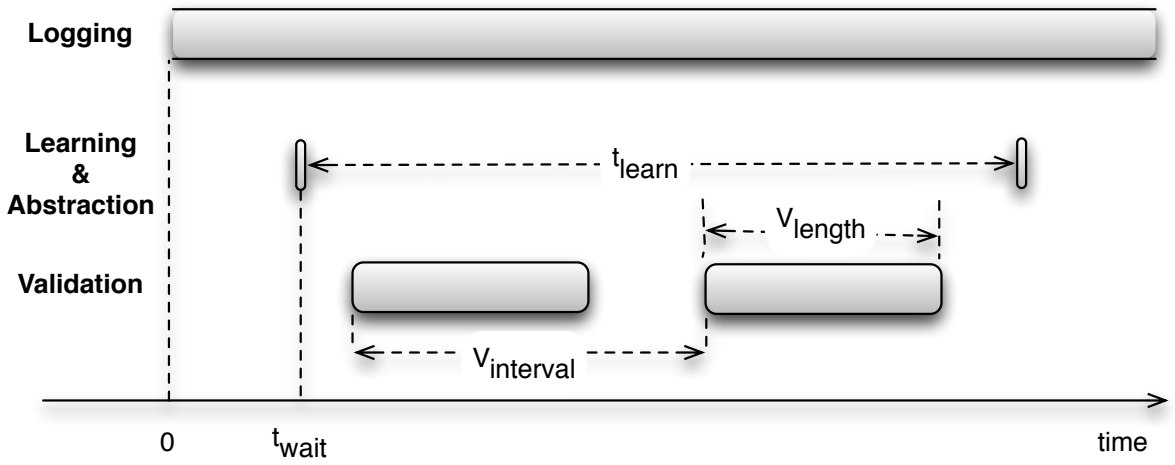


Figure 19.3: The three rules – **Log**, **Learning & Abstraction** and **Validation** – inside an *observer* are executed at specific time steps.

19.4 Adaptive Abstraction of Spatial Agents

The goal of the proposed abstraction is to adaptively reduce the number of agents – N in Equation (19.12) – during the course of the simulation. We immerse *observer* agents, or *observers*, in the simulation to monitor the agents and learn their adhesion patterns. *Observers* are defined the same way any other agent in the system is defined, i.e. through Equation (19.3) with only one behaviour component. The behaviour engine executes the rules in the *observers* at each time step during the course of the simulation.

Each *observer* has three rules in its behaviour component (Figure 19.3). An *observer* constantly monitors the simulation space and logs certain information about agents and their interactions. Once enough information is logged ($t > t_{wait}$), the *observer* tries to detect and learn an adhesion pattern that describes which agents have been sticking together. If this pattern is detected, the *observer* creates a meta-agent that subsumes other individuals or meta-agents. In order to validate the behaviour of their meta-agents, *observers* periodically check whether the deployment of the subsumed individual agents yields an outcome different from the predictions of the learned pattern. If the discrepancy between these two outcomes exceeds a given threshold τ_{conf}, the *observer* destroys the meta-agent and restores the subsumed individual agents. The process of learning and validating happens periodically resulting in an abstraction hierarchy that is adaptive over the course of a simulation.

Table 19.1 summarizes the conditions and actions for each rule. We describe each of the rules in Table 19.1 in the following subsections, and then we present a computation analysis of our abstraction mechanism.

Table 19.1: Three rules of each *observer* along with their condition and action.

Name	Condition	Action
Log	*always*	UpdateGraph: logs information about agents and their interactions
Learning & Abstraction	$(t > t_{wait})$ && $(t \bmod t_{learn})$	Abstract: detect patterns and create meta-agents
Validation	$V_{interval} \leq t \leq V_{interval} + V_{length}$	Validate: validate meta-agents

19.4.1 The First Rule: Log

Observers are configured to maintain an adhesion graph $G_{Adhesion} = (V, E)$ whose nodes are the agents in the simulation. An edge e_{ij} between two agents denotes the strength of their adhesion, i.e. the longer two agents stick together, the larger the weight of their edge w_{ij} is. At every time step, an *observer* loops through the agents it is monitoring and updates the adhesion graph based on Algorithm 4.

Algorithm 4 The Action: UpdateGraph($G = (V, E)$)

1: **for all** Agent ag^i **do**
2: $comp^i_{physics} = ag^i$.getDependency(PhysicsComponent);
3: $partners = comp^i_{physics}$.collisionPartners();
4:
5: **for all** Agent j **in** *partners* **do**
6: $w_{ij} \leftarrow w_{ij} + \Delta_{inc}$;
7: **end for**
8:
9: **for all** Agent j **in** V_i **do**
10: {V_i is the set of neighbours in G for ag^i}
11: **if** $(w_{ij} > 0)$ && $(j \notin partners)$ **then**
12: $w_{ij} \leftarrow w_{ij} - \Delta_{dec}$;
13: **end if**
14: **end for**
15: **end for**

UpdateGraph is the action of this rule that updates the adhesion graph. For each monitored agent ag^i, its sibling physics component $comp^i_{physics}$ is fetched (line 2) to get the *partners* it is colliding with. Then for each collision partner j, the value of the edge between the two agents is incremented by some value Δ_{inc} (line 6). There might be other agents that were colliding with ag^i in previous time steps which are not colliding any more at this time step. Therefore, their corresponding edge should be decremented by a larger number ($\Delta_{dec} > \Delta_{inc}$) to ensure that once two agents stop colliding, their corresponding edge will be quickly set to zero. This number could also depend on the current value of an edge, but for simplicity we set it to be a constant number.

19.4.2 The Second Rule: Learning & Abstraction

An *observer* maintains an adhesion graph of agents. At certain intervals (t_{learn}), the *observer* finds clusters of agents that have adhered to one another for a sufficiently long duration of time, and subsequently, creates a meta-agent that subsumes the individual agents in each of these clusters. Since the structure of the meta-agents is the same as that of the individual agents, the same process can be applied to meta-agents, thus creating abstraction hierarchies during the course of the simulation.

Abstract is the action of the **Learning & Abstraction** rule which is described in Algorithm 5. It first creates another graph G' by removing all the edges in the adhesion graph G whose weight is less than some threshold θ (line 1). In this new graph, an edge between two agents means that they have been sticking together for an adequately long time. In the next step, we find all the connected components in this graph. Each cluster of agents in a connected component will be subsumed by a meta-agent.

Figure 19.4(a) illustrates an example of learning in which there are five agents in the simulation space. Agents A, B, C, and D are colliding while agent E is detached. Figure 19.4(b) shows the adhesion graph of an *observer* that is monitoring this simulation. Assuming that θ is 100, the new graph G' is constructed by removing the edge between Agents C and D whose value is 35. In the next step (line 2), the connected component algorithm finds a cluster that contains more than one agent (Figure 19.4(c)). Subsequently, a meta-agent M is created and the adhesion graph G is updated to reflect the subsuming meta-agent (Figure 19.4(d)).

Algorithm 5 The Action: Abstract(G, θ)

```
1: G' = (V, {e_ij : ∀e_ij ∈ E s.t. e_ij > θ});
2: clusters = connectedComponents(G'); {each cluster: a set of agents}
3: for all Set<Agent> aCluster in clusters do
4:   if aCluster.size() ≤ 1 then
5:     continue;
6:   end if
7:   Agent meta_agent = new Agent();
8:   {composing the hierarchy of the meta_agent}
9:   TransformComponent meta_transform = new TransformComponent();
10:  PhysicsCompositeComp meta_body = new PhysicsCompositeComp();
11:  BehaviourComponent meta_behaviour = new BehaviourComponent();
12:  meta_agent.add(meta_transform);
13:  meta_agent.add(meta_body);
14:  meta_agent.add(meta_behaviour);
15:  for all Agent anAgent in aCluster do
16:    meta_agent.add(anAgent);
17:    for all Component comp in anAgent do
18:      if isGraphicsComponent(comp) then
19:        continue; {nothing happens to the graphics component}
20:      end if
21:      if isPhysicsComponent(comp) then
22:        comp.active = false; {the physics component is disabled}
23:        {and attached to the composite meta_body}
24:        meta_body.attach(comp);
25:      end if
26:      if isTransformComponent(comp) then
27:        meta_transform.origin += comp.origin;
28:        comp.makeRelativeTo(meta_transform);
29:      end if
30:      if isBehaviourComponent(comp) then
31:        comp.active = false;
32:        for all Rule r in comp do
33:          if meta_behaviour.contains(r) == false then
34:            meta_behaviour.add(r);
35:          end if
36:        end for
37:      end if
38:    end for
39:  end for
40:  meta_transform.origin /= aCluster.size(); {making the average}
41:  {the addition of meta_agent will be reflected in G, c.r. Fig. 19.4(d)}
42:  update(G, meta_agent);
43:  state = waitToValidate; {setting up the validation mechanism}
44:  conf_initial = 50%;
45: end for
```

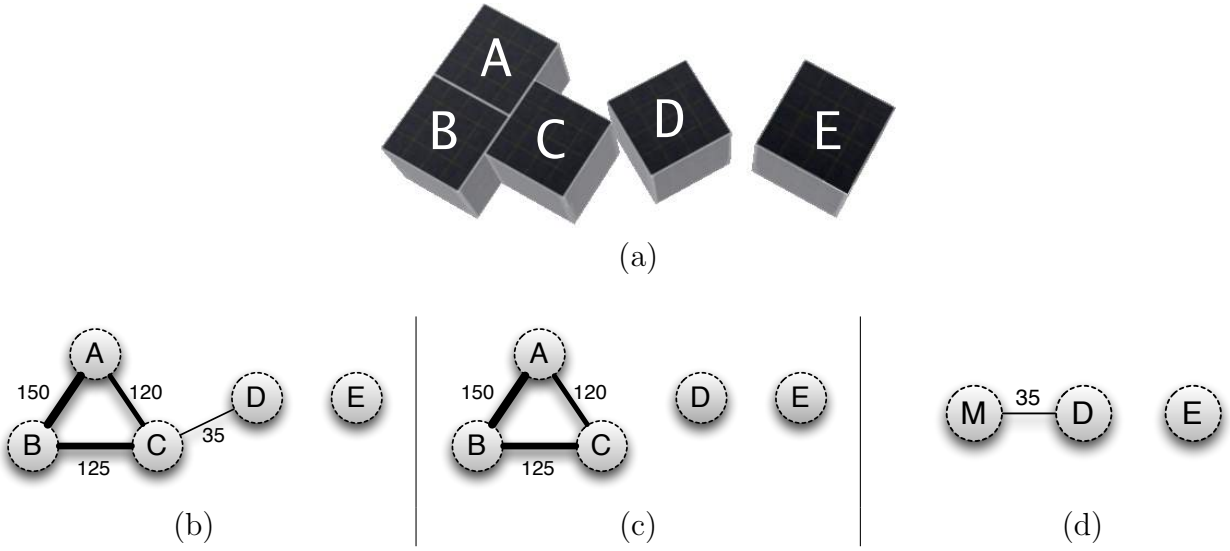


Figure 19.4: (a) Five agents in the simulation space. (b) The adhesion graph G maintained by the observer. (c) The modified graph G' in which weaker links ($w_{ij} < 100$) are removed. (d) The new adhesion graph G is constructed by replacing agents A, B, and C with the new abstract agent M and restoring the previous connections in the old adhesion graph.

The components of the new meta-agent are configured as follows:

1. The behaviour component is the aggregation of all the unique rules in the subsumed agents.

2. A physics composite component encompasses individual physics components. From now on, the physics engine calculates the forces on this composite structure instead of the individual structures.

3. The origin of the transform component is the average origin of all the subsumed agents. Since the graphics component depends on a single transform component as its sibling, we do not disable individual transform components. In addition, all the transform components in the subsumed agents will become relative to the meta-agent's transform component to ensure proper movement of all the sub-parts when the meta-agent moves.

Figure 19.5 shows the representative data structures of the three subsumed agents in Figure 19.4, along with the structure of the meta-agent. Since the graphics engine requires that only one transform component be present as the sibling of each graphics component, the meta-agent also maintains each parent individual agent along with its transform and graphics components. The efficiency gains are a result of aggregating the rules in the behaviour component of each

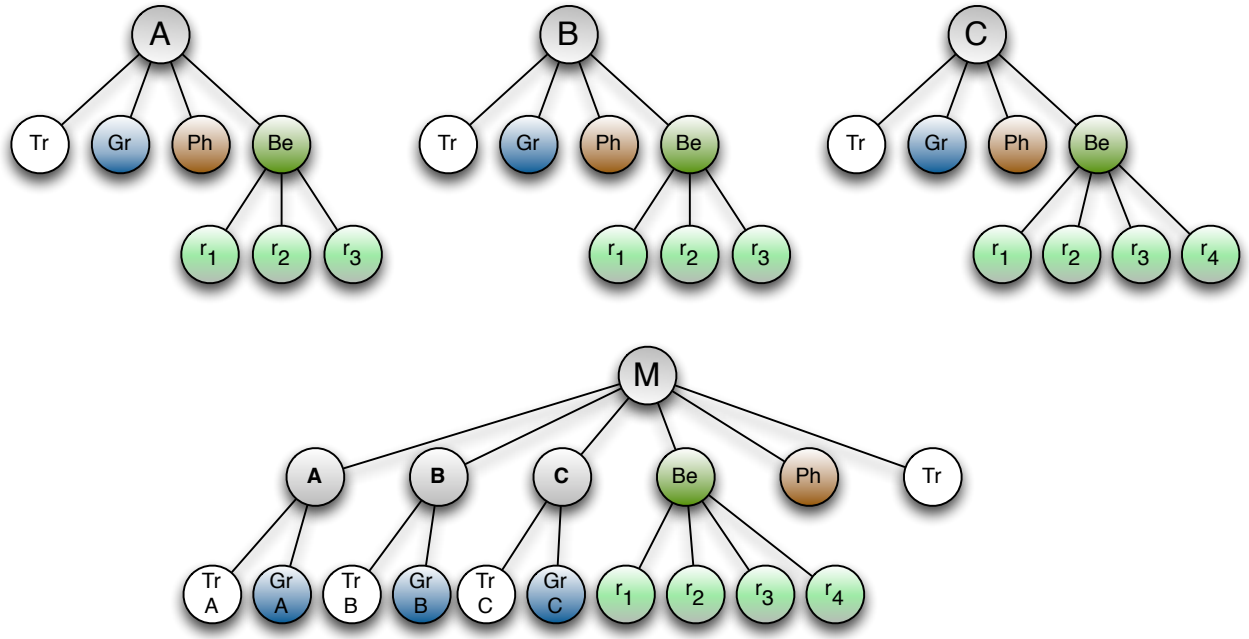


Figure 19.5: Three agents A, B, and C are subsumed by a meta-agent M. The meta-agent aggregates unique rules in its behaviour component resulting in a reduction of 10 individual rules to 4 rules in M.

individual agent into the meta-agent such that only the unique rules are added to the meta-agent.

The newly created meta-agents have a behaviour and a physics component, which enable them to undergo physical interactions as a whole, and also to execute their rules at each time step. This scale-free representation of meta-agents allows for further abstractions, as meta-agents are not any different from individual agents, and therefore they can be abstracted in the same way. For example, Figure 19.6 shows the next learning cycle in which agents D and M form the next meta-agent N.

An important consideration is to make sure that meta-agents show valid behaviours, as the dynamics of the system might change and individual agents might not stick together any longer. In this case, a validation mechanism should be in place to ensure that the meta-agent is destroyed and the individual agents are returned to the simulation. To this end, the *observer* sets up the starting state for the validation phase at the end of Algorithm 5. Also, it assigns an unbiased confidence value ($conf_{\text{initial}} = 50\%$) to the learned hierarchy (line 44). The validation mechanism is discussed in the next section.

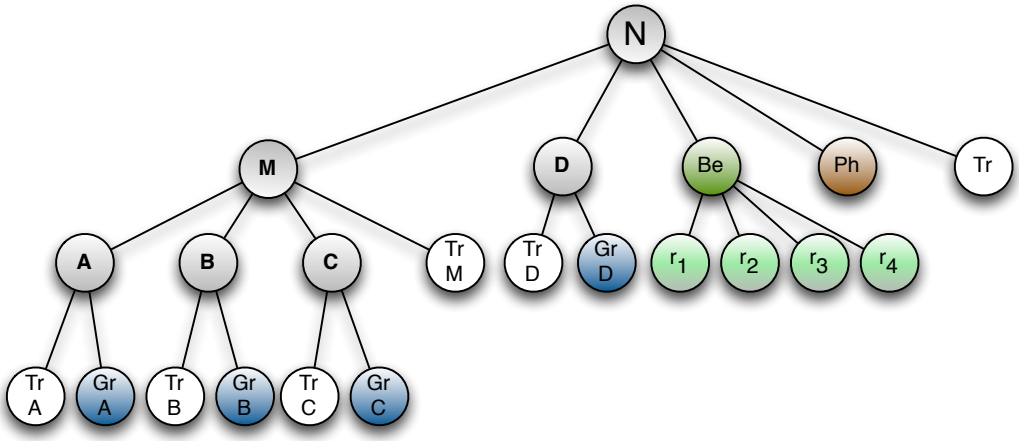


Figure 19.6: Assuming that agent D has the same structure as that of agent A, agents D and M form the new meta-agent N in the next learning cycle.

19.4.3 The Third Rule: Validation

After some time, a learned hierarchy might not be valid any more. The *observer* needs to ensure that a learned hierarchy is valid to be simulated. As the abstraction is an online process taking place as the simulation proceeds, there is no future expected data to conduct a supervised validation algorithm. As a result, the *observer* has to rely on unsupervised measures to validate a learned pattern. One such measure is the discrepancy between the outcome of the expected behaviour and the deployed behaviour of a learned hierarchy.

Algorithm 6 shows the proposed validation mechanism. At regular time intervals V_{length}, the *observer* releases a subset of the abstracted agents back into the simulation (line 6). After the validation period V_{length}, the *observer* re-abstracts those test agents (line 13) and regulates the confidence value (line 16) based on the difference between the behaviour of the test agents compared to the behaviour expected by the *observer*. In particular, if the individual test agents stick together in the validation period, the confidence value will be increased. A confidence measure below a given threshold indicates that a learned hierarchy is not valid any longer and that the *observer* has to break the learned abstraction by releasing its subsumed agents into the simulation (line 20).

Algorithm 6 The validation algorithm

```
1: {t is the simulation time step}
2: if (state == waitToValidate) && (t mod V_interval == 0) then
3:    state = validating;
4:
5:    {undo the abstraction for a subset of abstracted agents}
6:    testAgents = undoAbstraction(V_ratio);
7: end if
8:
9: if (state == validating) && (t mod (V_interval + V_length) == 0) then
10:   state = waitToValidate;
11:
12:   {re-abstract test individuals}
13:   reAbstract(testAgents);
14:
15:   {regulate the confidence value based on the performance of individuals against what was
      expected}
16:   conf = regulate();
17:
18:   {if the confidence is less than a threshold, break down the learned abstraction}
19:   if conf < τ_conf then
20:      breakAbstraction();
21:      state = noValidation;
22:   end if
23:
24: end if
```

19.4.4 Computational Analysis of the Proposed Abstraction Mechanism

The introduction of *observers* adds an overhead to the run-time of the simulation. On the other hand, a successful abstraction should reduce the run-time shown in Equation (19.12). Therefore, an analysis to identify the parameters of the abstraction is necessary. To this end, we define the run-time of a simulation in the presence of an *observer* as follows:

$$Step^t = N' * R + Step^t(log) + Step^t(learn) + Step^t(validate) \tag{19.13}$$

$$N' = \alpha N + M \tag{19.14}$$

where N' is the number of agents in the simulation, α is the percentage of the unsubsumed agents and M is the number of meta-agents. Ideally, we want to abstract as many agents as

possible ($\alpha \to 0\%$) into a single meta-agent ($M = 1$).

Calculating the timing for the first rule is straight-forward. Since each individual agent has a bounded number of collision partners, the inner loops in Algorithm 4 are executed in constant time and therefore, $Step^t(log)$ is equal to the number of agents in the system, i.e. $Step^t(log) = N'$.

The performance of the second rule depends on how the adhesion graph is implemented. Generally, finding connected components in a graph $G = (V, E)$ requires $O(|V| + |E|)$ where $|V|$ is the number of nodes in the graph, i.e. N' in the adhesion graph. Assuming that the addition of the unique rules in the behaviour component of a meta agent requires $O(R)$, the time required to execute the second rule is calculated as follows:

$$Step^t(learn) = \begin{cases} N' + |E| + N' * R & \text{if } (t \mod t_{learn}) == 0 \\ 0 & otherwise \end{cases} \tag{19.15}$$

where $|E|$ is the number of the edges in the adhesion graph and R is the maximum number of interaction rules in a behaviour component.

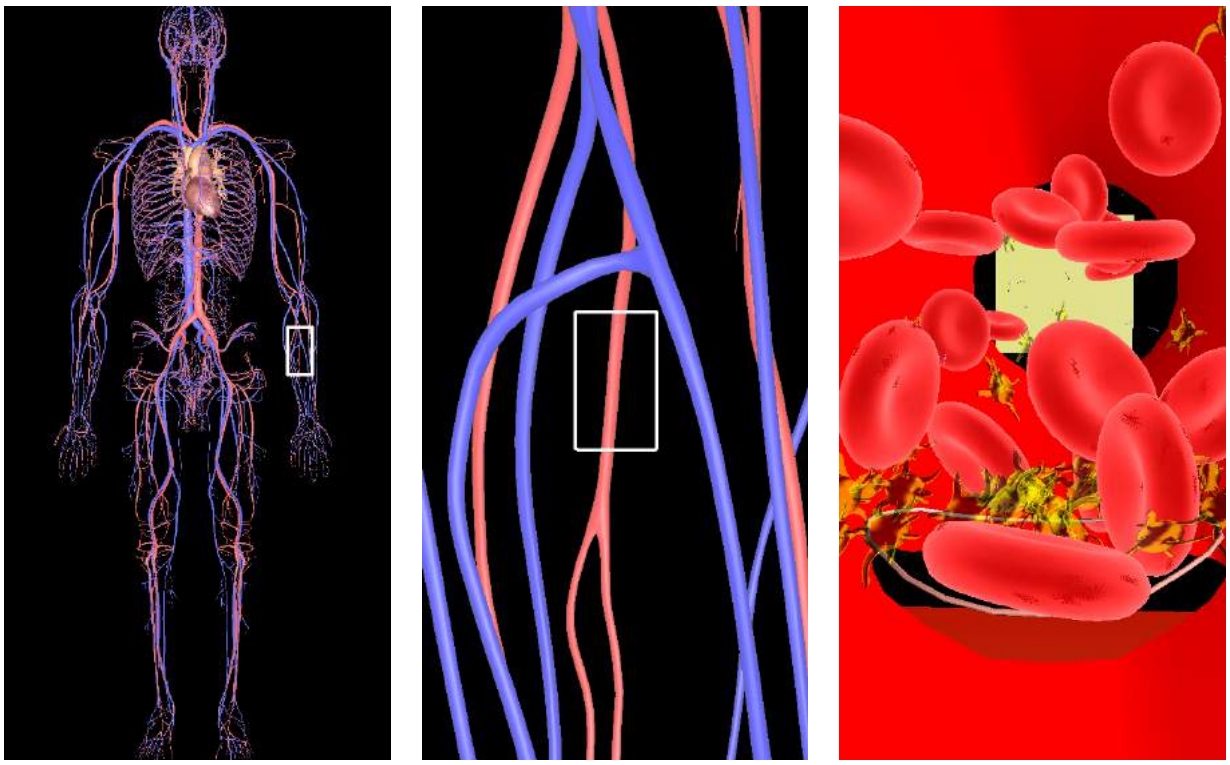
The last rule – **validation** – simply involves monitoring a subset of subsumed agents in meta-agents and requires the following time:

$$Step^t(validate) = \begin{cases} V_{ratio}(1-\alpha)N & \text{if } V_{interval} \leq t \leq V_{interval} + V_{length} \\ 0 & otherwise \end{cases} \tag{19.16}$$

where V_{ratio} is the percentage of the subsumed agents whose original structure is restored in the validation cycle.

19.5 Experiments

The proposed self-organized learning and abstraction method can be employed in any agent-based simulation in which individual agents form groups of agents by sticking together spatially. Biological simulations are particularly suitable applications as biological entities are formed from the aggregation of smaller entities. Our agent-based framework, *LINDSAY Composer* is a part of *LINDSAY Virtual Human* [540] – a 3-dimensional model of human anatomy and



(a) (b) (c)

Figure 19.7: The blood coagulation simulation is a part of a family of simulations, designed to study the circulatory system in *LINDSAY Virtual Human*.

physiology. Blood coagulation is one of the early simulations implemented in this framework, which belongs to a family of simulations to study the circulatory system (Fig. 19.7). While explaining the circulatory system and its simulation is beyond the scope of this paper, we applied our proposed method to an agent-based simulation of blood coagulation, which will be described in the next subsection.

19.5.1 Model Setup

Blood coagulates at wound sites because of the interplay of various bio-agents such as platelets, fibrinogens, and serotonins. If a collagen protein around the wound site collides with a platelet, the platelet becomes activated. In case that an activated platelet collides with the wound site, it secretes several chemicals which in turn activate more platelets in the blood vessel. Gradually, a network of fibrinogens together with a platelet plug form a clot around the wound site, as shown in Fig. 19.7(c)[1].

[1]The blood coagulation simulation in this paper extends the experiment reported in [516]. It also lists all the agent structures, along with their rules and parameters. The time it takes to form a clot is different in each

Table 19.2: Agent description

Agent Types	**Description**
Platelet, Fibrinogen, Serotonin, and Thrombin	Their interaction results in the formation of the clot.
Red and White blood cells	They participate in the formation of the clot by getting stuck in the wound site.
Destructor	Removes the agents it is colliding with from the simulation.
Emitter	Adds new agents into the simulation space and positions them randomly in a pre-defined volume.
Flow field	Applies a fluid flow force onto the agents thus moving them along a given direction.
Blood Vessel	Defines a volume in which the flow fields move other agents.
Wound	The wound site that interacts with platelets.

We identified eleven agents for this simulation, as listed in Table 19.2. Figure 19.8 shows the initial setup of the agents that exist at $t = 0$. The emitter agent produces platelets and fibrinogens and randomly positions them in a small volume at the right side of the blood vessel. A horizontal flow field moves all the platelets and fibrinogens along the blood vessel. There is a vertical flow field that pushes the agents to exit through the wound hole. Consequently, the agents exit the blood vessel either through the wound or once they reach the end of the blood vessel. Once the agents exit the blood vessel, they are no longer needed in the simulation and removed by the two destructors at both exits.

Most of the agents in Table 19.2 have a behaviour component consisting of a set of rules. Agents can share some rules while at the same time having their own unique rules. Although thrombin and serotonin agents, and also red and white blood cell agents share the same behaviour rules, they collide with other agents in different ways, since their physical structures are different. In the following, we describe the rules for each agent:

Platelet

r_1: **Self Activation**

experiment as a result of different values of the parameters, which makes them non-identical.

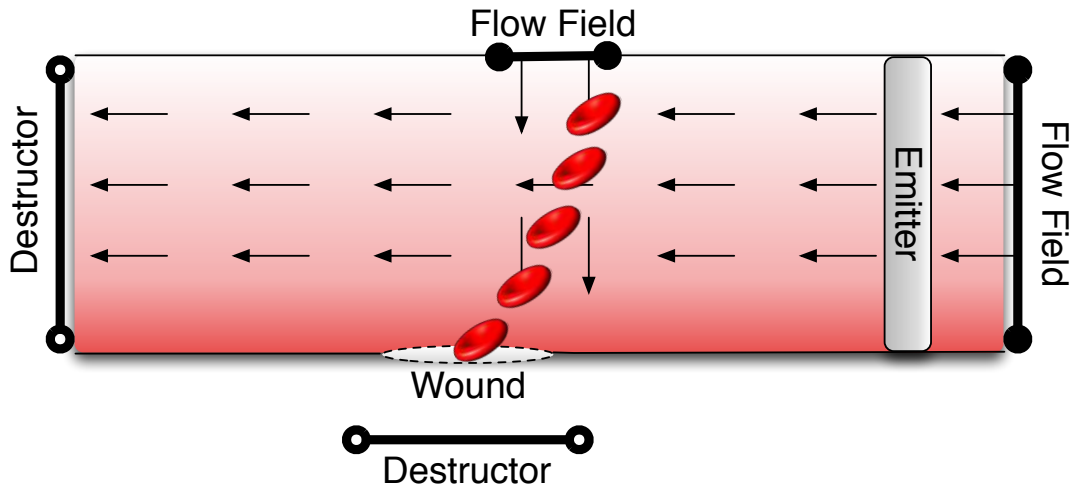


Figure 19.8: The simulation state at $t = 0$, the emitter agent produces platelets and fibrinogens which are moved by the flow fields inside the blood vessel. There is a hole in the wound site through which some agents exit the blood vessel. Two destructors remove agents that are not needed any longer.

if (agent is deactivated) **AND** (agent is colliding with either an activated platelet or the wound) **then** activate the agent

r_2: **Fibrinogen Activation**

if (agent is activated) **AND** (agent is colliding with a deactivated fibrinogen) **then** activate the colliding fibrinogen

r_3: **Adhesion-1**

if ($mass > 0$) **AND** (agent is activated) **AND** (agent is colliding the wound) **then** set $mass$ to 0

r_4: **Adhesion-2**

if ($mass > 0$) **AND** (agent is activated) **AND** (agent is colliding with an activated platelet or an activated fibrinogen) **then** set $mass$ to 0

r_5: **Secretion**

if (agent is activated) **AND** ($rand() > 3\%$) **then** secrete randomly a new thrombin or serotonin

r_6: **Random Walk**

if `(TRUE)` **then** `random walk in the space`

Fibrinogen

r_1: **Self Activation**: same as r_1 in Platelet

r_2: **Adhesion-1**: same as r_3 in Platelet

r_3: **Adhesion-2**: same as r_4 in Platelet

r_4: **Random Walk**: same as r_6 in Platelet

Thrombin and Serotonin

r_1: **Chase**

if `(TRUE)` **then** `accelerate toward a randomly selected, deactivated platelet`

r_2: **Random Walk** same as r_6 in Platelet

Destructor

r_1: **Destruct**

if `(agent is colliding with another agent)` **then** `remove the colliding agent from the simulation`

Emitter

r_1: **Generate**

if $(t \mod 15)$ **then** `generate 2 platelet and 2 fibrinogen agents with random positions`

Flow field

r_1: **Move**

if `(TRUE)` **then** `apply a physical force on all the agents inside the given volume`

Red and White Blood Cell

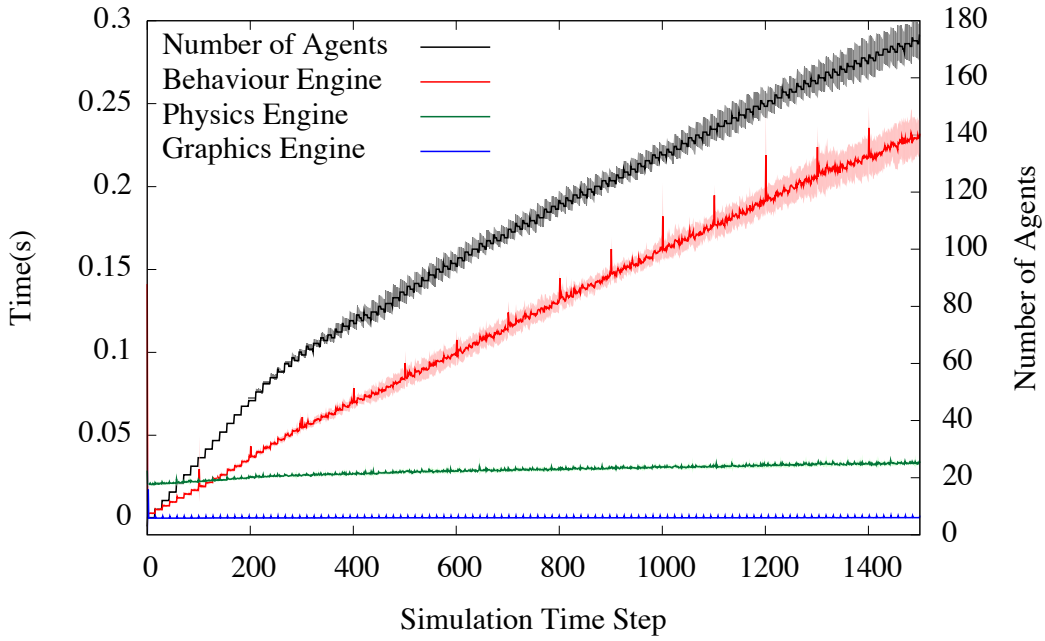


Figure 19.9: Run-times of the three engines along with the number of agents per simulation time step. The run-time of the graphics engine is almost at zero.

r_1: **Random Walk**: same as r_6 in Platelet

We ran the simulation for 1500 time steps 10 times. Each simulation started with 3 agents and ended with nearly 180 agents. Figure 19.9 shows the run-time of the three engines in the simulation. It confirms our previous claim that the behaviour engine is the bottleneck of the simulation. The physics and the graphics engine have a constant run-time independent of the number of the agents while the run-time of the behaviour engine grows approximately linearly with the number of agents. The linear growth of the run-time of the behaviour engine stems from the fact that none of the rules actually search in the list of the agents, hence it follows the asymptotic complexity of $O(N * R)$, as explained in Section 19.3.

19.5.2 Observer Setup

In addition to the individual agents, we add one *observer* to the simulation. Table 19.3 lists all the important parameters in our system. The *observer* monitors the simulation space and updates the adhesion graph based on Algorithm 4. After the *observer* monitors the simulation long enough (t_{wait}), at specific intervals (t_{learn}) it finds the connected components and subsequently, creates the meta-agents. The meta-agents subsume the individual agents according

Table 19.3: System parameters

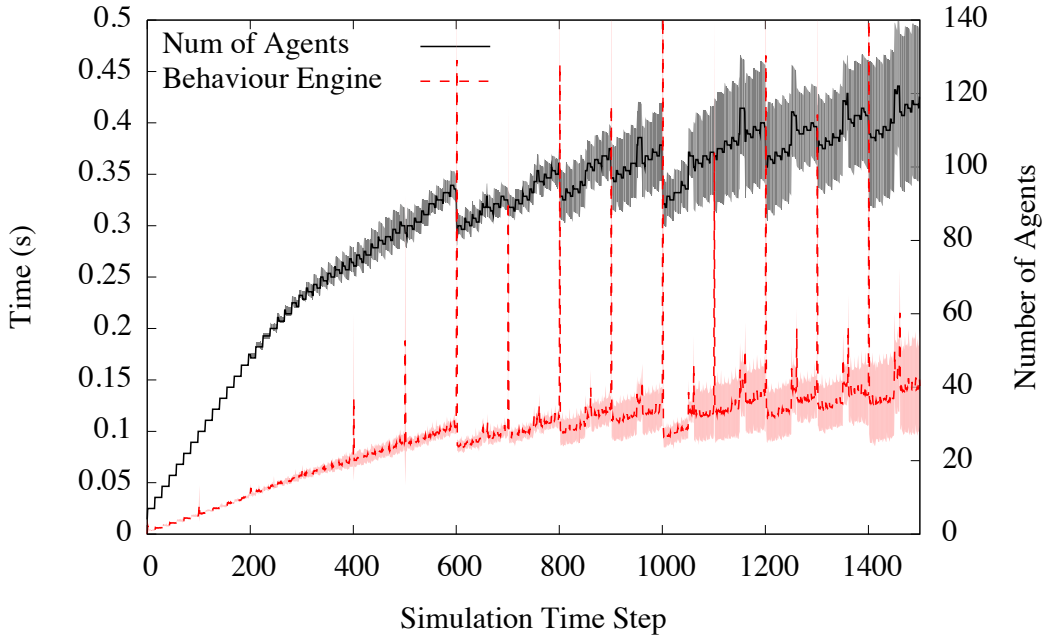
Parameter Name	Symbol	Value
Adhesion incremental weight	Δ_{inc}	1
Adhesion decremental weight	Δ_{dec}	5
Delay before learning	t_{wait}	400
Learning interval	t_{learn}	100
Cutoff threshold for the adhesion graph	θ	200
Validation interval	$V_{interval}$	50
Validation length	V_{length}	10
Validation ratio	V_{ratio}	10%
Confidence threshold	τ_{conf}	0.4

to Algorithm 5. In predefined intervals, $V_{interval}$, the *observer* randomly chooses a subset of the subsumed individuals in every meta-agent and restores their original hierarchy. The size of this subset is determined by V_{ratio}. After some time, V_{length}, the *observer* puts the individual agents back in the subsuming meta-agent and validates its abstractions based on the resulting interactions compared to the expected result. The confidence of the learned pattern is regulated accordingly. If the confidence of a pattern is less than some threshold τ_{conf}, the according meta-agent will be removed and its subsumed agents will be put back in the simulation.

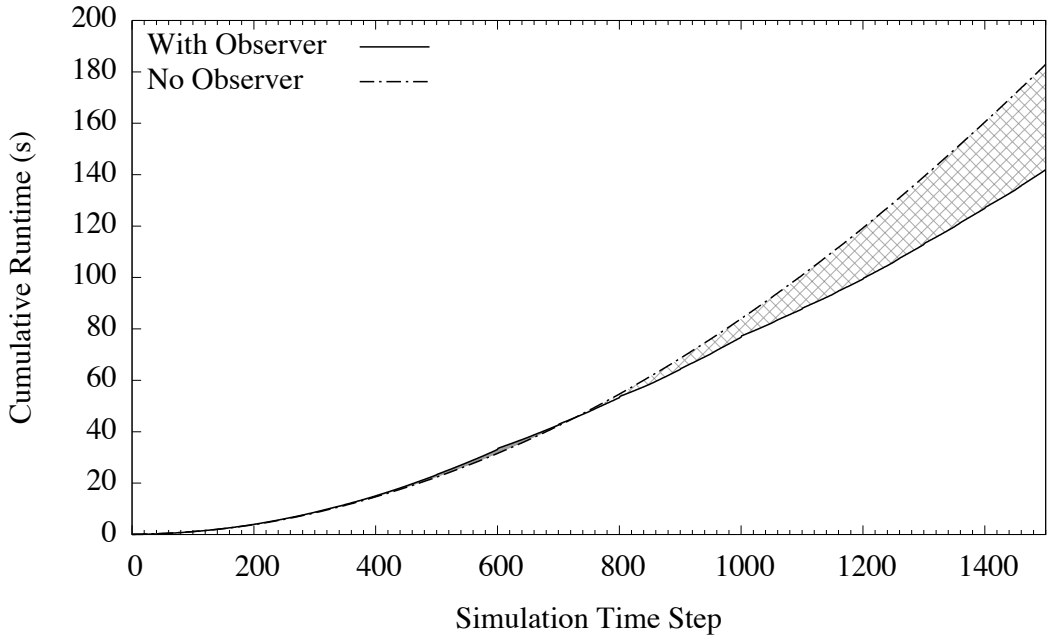
19.5.3 Results

Figure 19.10(a) shows the run-time of the behaviour engine in the presence of the abstraction mechanism. Compared to the normal run of the simulation (Figure 19.9) in which there are almost 180 agents at the end of the simulation, the abstraction mechanism reduces this number to 120. Speeding up the simulation is the immediate result of this abstraction. The larger peaks in Figure 19.10(a) denote the learning intervals (t_{learn}) while the smaller peaks happen at the validation phase (V_{length}). Figure 19.10(b) depicts the cumulative run-time of the simulation comparing a normal run against a run with the *observer*. Adding the *observer* introduces no measured overhead while at the same time reducing the total run-time of the simulation from 180 seconds to 140 seconds resulting in a 20% reduction of the run-time.

Figure 19.11 shows the agent adhesion graph in a sample run at $t = 900$, in which five connected components are distinctive by their colours. There are 37 agents in the biggest cluster (enclosed



(a)



(b)

Figure 19.10: Run-time with and without the *observer*, (a) Run-time per simulation time step, (b) Cumulative run-time.

by a dashed line) consisting of 16 platelets, 9 fibrinogens, 9 red blood cells, and 3 meta agents. Together, they have $16*6+9*4+9*1+3*6 = 159$ rules. On the other hand, the resulting meta-agent will only have 6 rules, as fibrinogens and red blood cells share the same rules defined in a platelet. Therefore, creating a new meta-agent will reduce the number of rules to be checked by 153. This reduction in the number of rules is mainly responsible for speeding up the simulation.

To verify that the abstraction mechanism produces the same or a similar behaviour as that of a normal simulation, we studied how the clot is formed during the course of the simulation. The clot concentration simply measures how many platelets, fibrinogens, or red blood cells are attached to the wound. We compare the result of ten normal runs of the simulation against ten runs of the simulation with the abstraction and report the result in Figure 19.12. This result suggests that the choice of values for the parameters resulted in the same system behaviour while at the same time speeding up the simulation.

To further study the validation mechanism, we introduced an abrupt change in the behaviour of the simulation at $t = 1000$, when we dissolve the clot by detaching the agents from the wound. As a result, there will be almost no platelet, fibrinogen, or red blood cell attached to the wound at $t = 1200$. We undo this new change at $t = 1500$ to let the clot form again. Figure 19.13 compares the behaviour of our proposed abstraction mechanism with that of the original simulation. The validation mechanism ensures that the system behaviour will adapt to the changes in the simulation – turning the validation off would result in an inaccurate system behaviour.

19.6 Discussion and Conclusion

We introduced the concept of abstraction to boost the speed of agent-based simulations by means of a light-weight *observer* agent that monitors the simulation space and abstracts groups of individual agents to higher-order meta-agents which in turn are subject to further abstractions. While the notion of *observers*, along with the steps to do an abstraction are shared between this work and our previous work [516], there are substantial differences between them:

1. The agent framework is explicitly defined in this work. This enables us to find the computational complexity of our agent-based simulations, with or without an abstraction

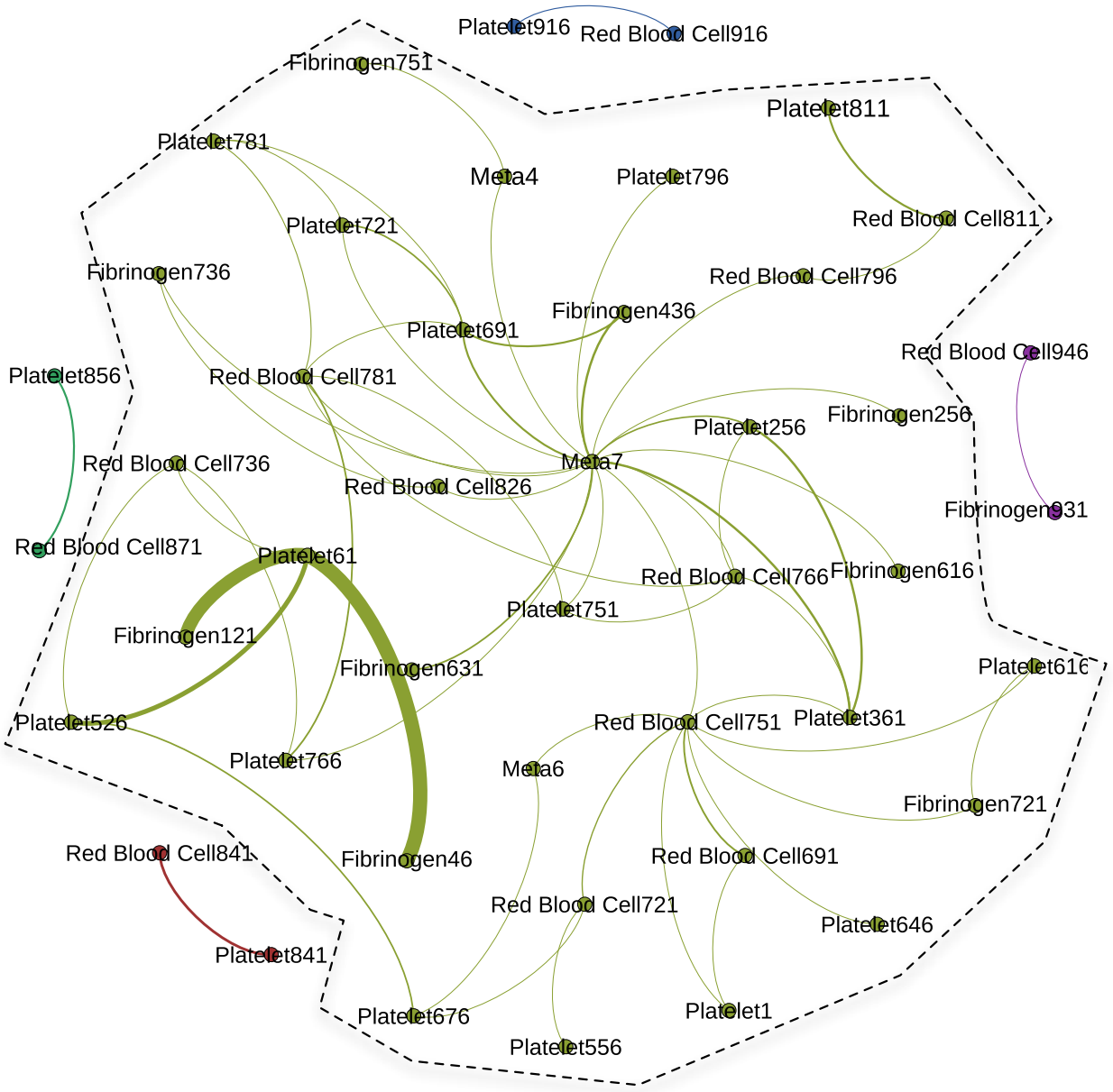


Figure 19.11: The agent adhesion graph in a sample run at $t = 900$ in which there are 5 clusters of connected components, in which the biggest cluster is enclosed by a dashed line. The weight of an edge between two nodes denotes the strength of their adhesion.

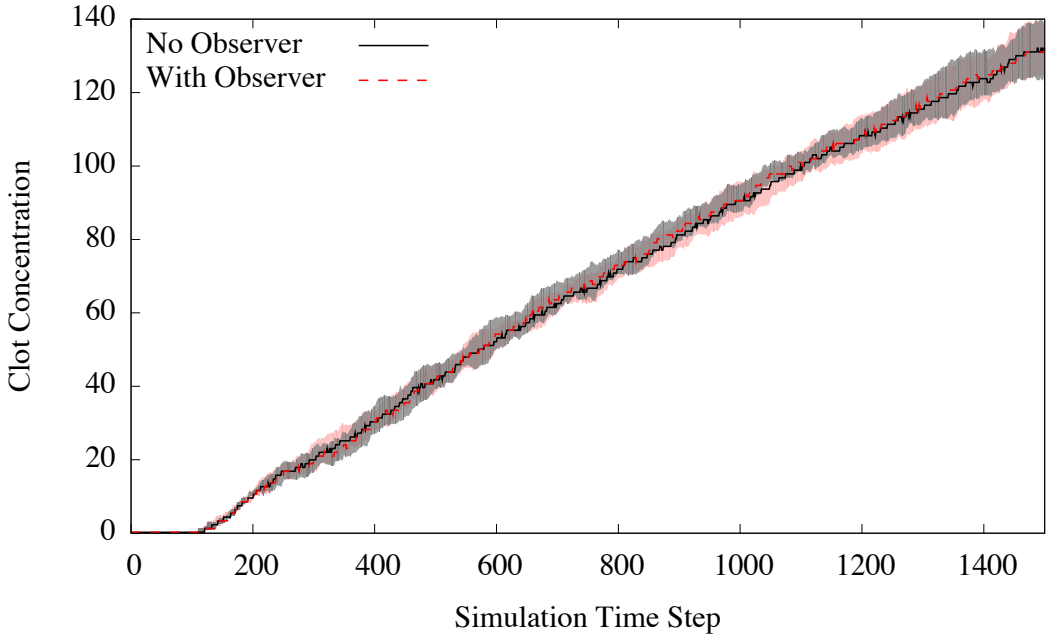


Figure 19.12: System behaviour in terms of the clot concentration, i.e. the number of platelets, fibrinogens, and red blood cells around the wound, reported over ten runs of the simulation with and without the proposed abstraction mechanism.

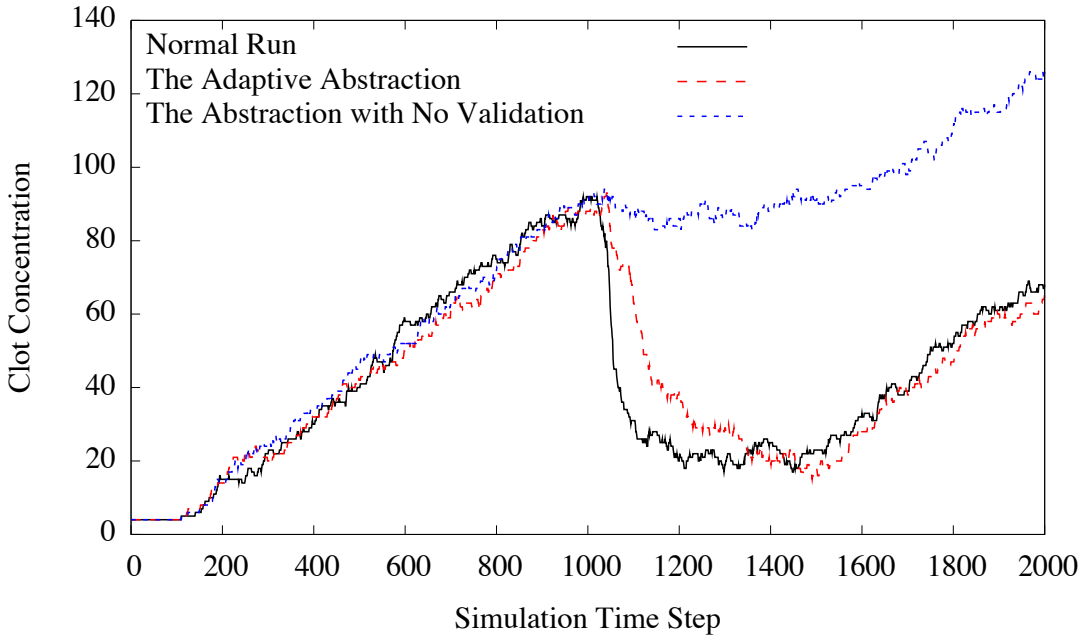


Figure 19.13: System behaviour in terms of the clot concentration. At $1000 < t < 1500$, the clot is forced to dissolve. An adaptive abstraction successfully follows the behaviour of the original run while an abstraction without validation results in an inaccurate system behaviour.

mechanism.

2. The objective of the work presented here is to create agent hierarchies by constantly abstracting many individual agents to one meta-agent, which can go under the same abstraction. In contrast, the algorithm in [516] abstracts many individual rules into one meta-rule, which does not change the agent structure. More precisely, the goal of this newly proposed abstraction mechanism is to reduce N in Equation (19.12), while the goal of our previous abstraction was to reduce R in Equation (19.12).

3. The proposed abstraction in this paper only works on spatial agents. It uses the notion of proximity among agents as a heuristic indicator to abstract them. Our previous abstraction mechanism can work on any type of agents [516].

4. The *observers* in this paper maintain a directed graph of agents. The weight of an edge between two agents denotes their proximity strength. We apply a connected component algorithm in the learning phase to find a group of agents. The *observers* in [516] maintain a list of executed rules; they apply a k-means algorithm to find a dense cluster of rules whose parameters are constant. In the validation step, an *observer* in this work puts back a few abstracted agents in the simulation while an *observer* in [516] re-activates an individual rule in a few selected agents. Therefore, although both abstractions have three rules – **Log**, **Learning & Abstraction**, and **Validation** – their actual implementations are completely different.

Our proposed abstraction mechanism was applied to an agent-based simulation of blood coagulation, in which bio-agents stick together to form a clot around the wound site thus preventing further bleeding. We showed that the adaptive abstraction results in the same system behaviour but with a 20% faster run-time. We emphasized the role of our unsupervised validation algorithm to ensure the validity of meta-agents.

The proposed abstraction mechanism creates self-organized, dynamical hierarchies during the course of a simulation. Studying the emerging patterns in such hierarchies is of great interest, particularly in the case of biological simulations in which new entities at higher levels are formed as the result of interactions among lower level entities. For multi-scale modelling, a stable, higher order entity can be used in other time or spatial scales to manage the compu-

tational burden of the simulation. This could eliminate the need for exponential increases in computation power to model such systems.

Chapter 20

Bring it on, Complexity! Present and Future of self-organising Middle-out Abstraction

Sebastian von Mammen, Jan-Philipp Steghöfer. The computer after me: Awareness and self- awareness in autonomic systems, ch. *Bring it on, Complexity! Present and future of self-organising middle-out abstraction*, pp. 83–102, World Scientific Press, 2014.

20.1 The Great Complexity Challenge

The inherent complexity of many man-made or naturally occurring challenges—such as understanding the influence of human interference in ecosystems or interacting biological processes—is widely acknowledged. The ubiquitous networking paradigm has highlighted the elaborate webs of interactions and interdependencies between living beings, objects and processes. Yet we still lack an algorithmic framework capable of tackling the complexity of the world in terms of representation and computation. Thus, any step toward understanding—and predicting—the dynamics and emergent phase transitions of complex systems would greatly contribute to the advancement of science. Present-day societal challenges that could benefit from this kind of knowledge are plentiful, and can be found in fields ranging from the life sciences to economics

and engineering. To some extent, the mathematical analysis of complex systems can provide some insights about the phase transitions that may occur over time [643, 644]. However, this approach requires a great deal of effort and does not scale well, becoming intractable as the number of factors involved in a system increases.

What is more, the interactions that drive system transitions have to be identified and formalised a priori by the modeller. In contrast, an ideal model building process should require as little information as possible about a system's actual behaviour. It should be enough to only describe how the parts of a system interact, without building in any assumptions about when feedback cycles might be triggered to snowball into fundamental global system changes. In a model of this kind, the parts of the system that interact according to sets of internal rules (and so without any external, higher-level drivers of their collective behaviour) are known as 'agents'. Each agent in such a model is a self-contained entity with its own individually accessible data, states and behaviours. The sequences of interactions among agents and the traversal of their states in a computational simulation correspond to the emergent feedback cycles and phase transitions of complex systems. If we were able to detect patterns that are precursors to phase transitions and patterns that correspond to the system's global dynamics, we would automatically become aware of emergent phenomena.

Inspired by some of the grand ideas in artificial intelligence, machine learning, and artificial life, we present the SOMO (self-organised middle-out) algorithm, a concept that might contribute to the outlined quest. Its goal is dynamic abstraction, i.e. bottom-up learning given enough training examples and top-down validation to reaffirm or revoke the previously learned concepts. We take this opportunity to present the SOMO concept with an emphasis on its visionary aspects—how the idea could evolve from its most recent conception, its current implementation, towards that desirable, dreamed-about computer after me.

20.2 self-organising middle-out abstraction

Early 2011 we presented the self-organised middle-out (SOMO) concept [268], an approach that automatically builds abstractions bottom-up and validates and revokes them top-down—possibly both at the same time but in respect to different model aspects. As it works in both directions and as it bridges the gap between the orders of the model, it can be considered to

operate at the ‘ meso’ level of analysis.

Its foundation is an unsupervised learning method that observes and learns processes which occur—that is to say, emerge—during a computational simulation. A learned process pattern provides a shortcut to driving the evolution of the simulation. Instead of considering the series of all conditions that lead to the process' changes one step at a time, it suffices to recognise the emergence of the process. As a consequence, the detailed interactions are no longer executed but, whenever the according preconditions hold, the observed side effects are enacted in the system. Such automatically learned patterns may also be understood as abstracted process descriptions and they hold the promise of helping us to understand, explain, and compute complex phenomena in simple terms.

SOMO observes the simulation data and identifies process patterns, ‘biased’ only in terms of its representations (meaning that the way interaction patterns are represented by SOMO can influence the kinds of patterns that can be detected and so bias the result). The identified patterns are used to refine the computational model that drives the simulation process being observed. As the SOMO algorithm continues to observe and learn the patterns that emerge from the simulation, it continually increases the model's level of abstraction by introducing hierarchies of abstracted patterns. It is hoped that such hierarchies will to some extent coincide with the real-world conceptual boundaries that we identify in natural systems, such as the subdivision of the organisational complexity of animal anatomy into cells, tissues and organs. Since such abstractions are inevitably subject to noise and unknown conditions, we also introduce a confidence measure that is associated with each abstraction.

In the next section (Section 20.3), we present a variety of concepts that are both inspiring the SOMO algorithm and closely related to it. Section 20.4 introduces a (borrowed) example that nicely illustrates the emergence of high-order physiochemical compounds. Based on this example, we outline the SOMO concept in Section 20.5. Current SOMO implementations are explained in Section 20.6 and futuristic implementations around it are presented in Section 20.7. In Section 20.8, we conclude with a short vision about SOMO's potentials.

20.3 Optimising Graphics, Physics & AI

Various research interests and complementary research trends have been driving the design of the SOMO concept:

- There is the concept of *emergence* that tries to capture novel properties and descriptions of (sub-)systems of higher orders [270].
- There is the need for integrative approaches to *representing, modelling and simulating multi-scale systems*—this challenge is currently addressed by passing up and down value sets from separate, sometimes fundamentally disparate, model components [87, 269].
- And, there is the need for *abstraction*: a model so comprehensive as to span several degrees of scale, to host a large body of systems and subsystems, and to independently consider their intricate behaviours quickly outmatches the computing capacities of even the greatest of supercomputers.

Abstraction is not only the essence of model building in the first place but it is also the key to expressive and efficiently solvable models. We postulate that a model should be as detailed and as comprehensive as possible, while its (numeric) utilisation for the purpose of rather specific predictions or simulations should automatically lead to model simplifications and abstractions. Whenever possible, this should happen without jeopardising the model validity; whenever necessary, the loss of accuracy the abstractions cause should be made transparent. SOMO pursues this endeavour by building and maintaining hierarchies of abstractions learned from observation. The higher the level of hierarchy, the fewer interactions have to be tested. Such tests are typically intertwined with expensive condition queries—only the state changes of the simulation will be performed to drive its evolution.

Similar shortcuts by means of hierarchical organisation have been conceptualised and implemented in numerous other contexts. For instance, different levels of detail (LOD) of computer graphics resources such as meshes (differing in the numbers of vertices) and textures (differing in the numbers of pixels) are typically organised in hierarchies to allow for fast access to the most commonly used assets, whereas the graphics scenes themselves are often subjected to spa-

tial partitioning hierarchies that allow algorithms to quickly determine which graphics objects need to be rendered in a given view port [645].

There is a significant overlap between these *culling* techniques and mechanisms to speed-up the detection of collisions between geometric objects, one of the foundational functionalities of physics engines—both rely on the quick discovery of objects at specific locations. In general, the locations of the geometries may change, which is why the spatial partitioning hierarchies are dynamically created and adjusted. Dynamic adjustments of the bounding volume hierarchies are also required if the geometries themselves are dynamic, for instance if they change their scale. In this case, a method has been shown to yield rather good results that updates the upper half of the hierarchy bottom-up if one of the geometries changes. The lower half is only updated selectively in a top-down fashion, as soon as the changed geometry is accessed [509].

Hierarchical optimisations have also been deployed in the field of artificial intelligence. For example, costly automated planning routines can be pruned early, if high levels of a hierarchy reflect the adherence of a plan's most critical variables [646]. Similarly, reflective agents need to plan their coordination—hierarchical abstractions of their interaction partners may increase their decision performance, too [647].

20.4 Emergence and Hierarchies in a Natural System

In Rasmussen et al. [10], an approach, or " Ansatz", to capturing the emergence of physicochemical compound objects with according emergent properties is described. We want to use their example to illustrate the mechanics of SOMO. In their experiments, attracting, repelling, and bonding forces among charged monomers and water molecules are shown to result in higher-order polymer and micelle formations—at each level, the resulting compounds obtain novel physical and chemical properties. In the model, hydrophobic monomers bind to hydrophilic monomers as well as to polymerised hydrophobic monomers, which results in 2^{nd}-order amphiphilic polymers which, in turn, aggregate in 3^{rd}-order micelle structures. At each stage, the resultant compounds exhibit properties different from the underlying constituents; The aggregating nature of the process yields compounds of greater size but it also leads to varying qualitative, geometric structures and differentiated physiochemical behaviours. An adapted illustration of the emergent process is shown in Figure 20.1.

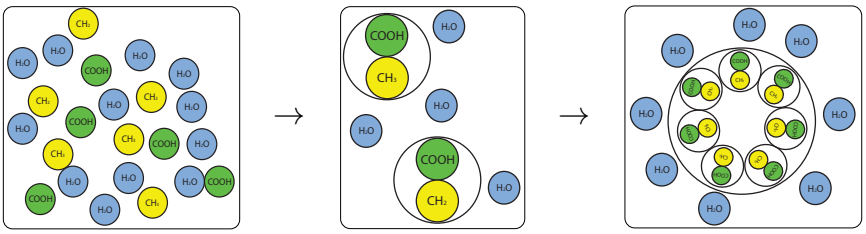


Figure 20.1: (a) Hydrophobic and hydrophilic monomers immersed in water. (b) Polymers emerge as hydrophobic monomers bind to hydrophilic monomers. (c) A micelle-like structure forms based on aligned polymers with hydrophobic heads and hydrophilic tails. These illustrations are adapted from [10].

The higher-order objects form based on the interactions and (emergent) relationships among the axiomatic objects of the given model. Often, higher-order objects can be captured as spatial aggregations but in general they should be regarded as networks of arbitrarily complex topologies. In accordance with [270], the authors also stress that emergent characteristics of a (sub-)system are observable in terms of its interactions.

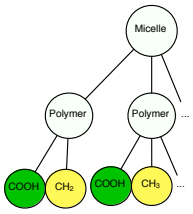


Figure 20.2: Order hierarchy.

We reflect the subsumption of individual elements by emergent entities of greater order in a hierarchical structure. In the given case, polymers are built from monomers and aggregate to form micelle-like structures (20.2). As Rasmussen et al. [10] suggest, an observer needs to identify the emerging units and their emergent properties; In our approach such observers are immersed in the simulation and observe the state and interaction patterns of the model entities. The observers further simplify the entities' computational representations in accordance with the learned behavioural patterns. Individual entities and their behaviours are subsumed by higher order entities that perform the learned patterns only in order to prune the computational complexity. However, we do not postulate a *necessary* coincidence between the learned high-order entities and emergent entities that we ourselves would identify, as in the micelle-example. Rather, we assume that there is a great chance that the learned patterns and the ones recognised by humans overlap to some extent—it is possible that the human-identified orders represent all but a small fraction of the automatically generated abstractions. In order to clarify this distinction, we step through an exemplary run of the SOMO algorithm in the next section, using the self-assembly of micelles as a running example.

20.5 The Technical Concept of SOMO

For our approach, we consider the elements of a model agents, described by their states and behaviours (for a more in-depth formalisation of the agent concept, consider, for instance, Denzinger's generic agent definition [55]). In our example, we distinguish between freely moving reactive agents that represent molecular compounds (similar to artificial chemistries [394]) and the environment they are immersed in. In particular, in our running example, a large number of hydrophobic and hydrophilic monomers is immersed in an aqueous environment. With the beginning of the simulation, the agents start to interact with each other and with the environment based on their behavioural rules. Together with the initial configuration of the system, these rules determine the result of the simulation, and if correctly phrased, they would result in the emergent phenomena described in the previous section.

20.5.1 Observation of Interactions

In addition to the model agents comprising molecular compounds and the environment, the SOMO concept introduces *observer* agents that monitor the interactions of the model agents as well as the conditions under which they occur. In the context of the micelle-forming example, observers do not have to make assumptions about the model agents' internal states and behaviours—only their actually triggered, externally observable state boundaries (i.e., the observed boundaries of the domain over which the state variable is defined) and state changes are relevant. However, the potency of the observers can be increased by granting them access to the agents' behavioural rule sets, to their internal states, and, thus, to their activated rules[1]. Following the fundamental concept of cause and effect, the observed interactions are recorded in terms of states and state changes. States that lead to certain state changes are translated into boundary conditions, or *predicates*, whereas state changes simple describe the transition from one state attribute value to another. Boundary conditions of time (i.e., the agents' timing), proximity between agents, or their mere presence or absence come to mind. Conjointly occurring pairs of boundary conditions and state changes are stored in *interaction histories*

[1]Focusing on the observation of state changes deems simpler than considering the underlying, responsible behavioural representations, as those would have to be correctly interpreted and related to the simulation context by an external observer.

over a certain period of time. A sliding time window reduces the storage required and lets the observers "forget" rare or singular events.

In the example, a pair of hydrophilic and hydrophobic monomers may attract each other, then stick together. A strong correlation between their locations would emerge, quickly resulting in a static relationship between their position states. The molecules might stick together over a long period of time. An observer would identify this behaviour and infer from the observations that these molecules will, under the given conditions, continue to stick together. Therefore, instead of continuously adjusting their locations based on their proximities at each time step of the simulation, an abstraction is introduced into the model: For now, the monomers are considered constantly attracted, or *bonded*. These bonded monomers, or *polymers*, are likely to aggregate in a *micelle*-like organisation because of the interplay with the aqueous environment: The polymers' heads align to face the water molecules, whereas their tails avoid them. Again, this formation is recognised and learned by the SOMO observers.

Instead of using specialised observers, the agents that make up the model can themselves observe interactions and the environment. In many cases, however, it is desirable to separate SOMO logic and the simulation model to maintain a clear distinction between the behaviours of the automatically learned abstractions and the original model. Independent of the kind of agent that takes on the task of observation and abstraction, the observers are subjected to certain restrictions. First, they are subject to an " event horizon", i.e., they do not perceive the entire system but only portions of it. This is due to the fact that an omniscient observer would have to deal with a vast amount of data, nullifying the scalability benefits SOMO was designed for and making it necessary to introduce limits of the observations. Second, even though observers make no assumptions about the model of an observed agent, they are limited to knowledge they have been granted access to—they can only perceive states and state changes they were designed to sense. Therefore, if interactions take place hidden from the observers, for instance direct messaging between agents based on hidden internal states, these interactions will not become part of the interaction history.

These restrictions bias the abstraction process. If the scope of the simulation is well-defined, these restrictions can be mitigated rather easily—the SOMO agents can be distributed across the interaction space to cover important " hot spots" and the system designer can ensure the agents' ability to observe all relevant states and state changes. For more ambitious projects,

however, it might be necessary to create a wide variety of observers, capable of identifying many different kinds of interactions.

Heterogeneous configurations are also possible. For instance, a subset of agents might be part of the original model and yet observe and abstract others, whereas the remainder of the agent population might be either model agents or observer agents. Naturally, hybrid agents, that play both roles, are useful, if an abstraction hierarchy is part of the model. In the following, in order to avoid additional case distinctions, we will only distinguish between (1) a strict separation between observer and model agents, and (2) the capacity of all agents to observe and abstract.

20.5.2 Interaction Pattern Recognition and Behavioural Abstraction

The entries of the interaction history not only comprise some anonymous information about states and subsequent state changes but they also reference the involved interaction partners. Similar to [648], we use the interaction histories as databases for finding patterns in the agents' interaction behaviours. Previously unknown patterns, or *motifs*, can be identified in time series relying on various techniques such as learning partial periodic patterns [649], applying efficient, heuristic search [650], online motif search [651], and even the identification of patterns of multiple resolutions [652]. Motif detection is adapted to interaction histories by assigning symbols, e.g., A or B, to specific log entries and finding patterns in the resulting strings, e.g., BBABCCBBABDA. In the given example BBAB is a motif candidate.

The recurring sequence of interactions contained in the motif as well as the conditions that are part of it can be the basis for a *behavioural abstraction*. If interactions are recognised repeatedly, they can be abstracted in several ways—most simply, the predicates are not always checked; in full glory, a complex sequence of interactions can be fully abstracted and only the aggregated side effects, i.e. the state changes, can be enacted in the system. Hence, a motif that provides comprehensive information about the interaction partners and the actual interactions, would allow to rewrite the agent rules as efficient sequences of unconditional instructions, with source and target agents readily in place.

A repeatedly occurring motif in the example system is the interaction between hydrophilic head and hydrophobic tail of a polymer. As the effect of this interaction stays the same

once the monomers have bonded, it is not necessary to check these conditions and calculate the result of the interaction any longer. An observer that has monitored this interaction can thus suspend the rules that cause the effect but rather enact it directly. Of course, such an intervention requires direct access to the agents' rule bases and might not be possible in some systems (cf. Section 20.5.5). Instead of suspending a specific agent's behavioural rules directly, it's possible to subsume the agent as a whole. The next section will shed more light on this approach of hierarchical agent subsumption.

20.5.3 Creating and Adjusting Hierarchies

The polymer formation from simpler monomers provides an example for an abstraction even more powerful than simplifying specific interaction rules: If the agents keep interacting in a predictive manner, among each other and with their environment, they can be subsumed by one *meta-agent* that takes their place and that exhibits their external behaviour without continuously (re-)evaluating the interactions of its constituting elements. Recursive subsumption of agents and meta-agents yields a hierarchy of ever more abstract meta-agents.

The formation of hierarchies can be implemented by means of a set of special operators. In order to establish a hierarchical relationship, an agent might *enter* another agent. Alternatively, it might be *adopted* by another agent. Both actions yield corresponding parent-child relationships between the two agents. Such a parent-child relationship is reverted by *raising* a child in the hierarchy.

Depending on whether the agents observe their own interaction histories or specialised observers are used in the system, different kinds of behaviour are possible:

- If an agent observes its own interaction history and detects that it constantly interacts with another agent (or a group of other agents), it can create a new agent, assign its own abstracted behaviour, enter this new agent and deactivate itself. The newly created higher order agent then adopts all other agents that formed the original behaviour, adding their abstracted behaviour to its own, and deactivating them as well.
- If specialised observers are deployed in the system, they create the meta-agents and assign the agents to be subsumed to them. The meta-agent then follows the same steps as above.

The end result in both cases is a meta-agent that behaves just like the group of agents to the outside but does not need to evaluate internal interactions. The polymer as well as the micelle are examples of structures that can be abstracted in this fashion. In fact, the micelle shows how a true hierarchy can form: in the course of the simulation, polymers form first, are detected by the observers, and abstracted. Then, the polymers form a micelle which is internally stable and behaves consistently towards its environment. It can thus again be detected and abstracted so that only interactions between the micelle and the water molecules have to be evaluated.

Repeated applications of these abstraction rules yield continuously growing hierarchies with increasingly simplified behaviours. At the same time, hierarchies are dissolved when no longer appropriate. For this purpose, meta-agents repeatedly check for validity of the abstraction they represent by checking whether the original predicates still hold or by temporarily disbanding the abstractions, checking for the occurence of the abstracted interactions and either re-abstracting or abandoning the abstraction.

The subsumption of agents and their behaviours closely resembles the concept of modularisation and crafting hierarchical code. Figure 20.3 shows an according visual programming perspective on agents, their behaviours and behavioural interrelations; individual operators (spheres) are recursively nested to allow for the hierarchical design of behavioural modules, whereas the connections between inputs and outputs (cones) determine the flow of information at each hierarchical level [653]. The realisation of this visual modelling language has, in parts, been motivated by the need of a generic, hierarchical representation of agent behaviours.

20.5.4 Confidence Measures

The identification of motifs in the interaction history as well as the decision to resolve a hierarchy are based on *confidence estimation*. There is a large body of work around confidence in statistics [654] and its effective standardisation for use in the natural sciences is a vivid research area [655]. Confidence measures are also used in computational models of trust [656]. The general idea is to estimate the probability that a pattern occurs based on its preceding frequency over a given period of time.

In SOMO, repeated observation of interaction patterns increases the confidence value. A sufficiently great confidence value leads to abstraction. The confidence value also determines the

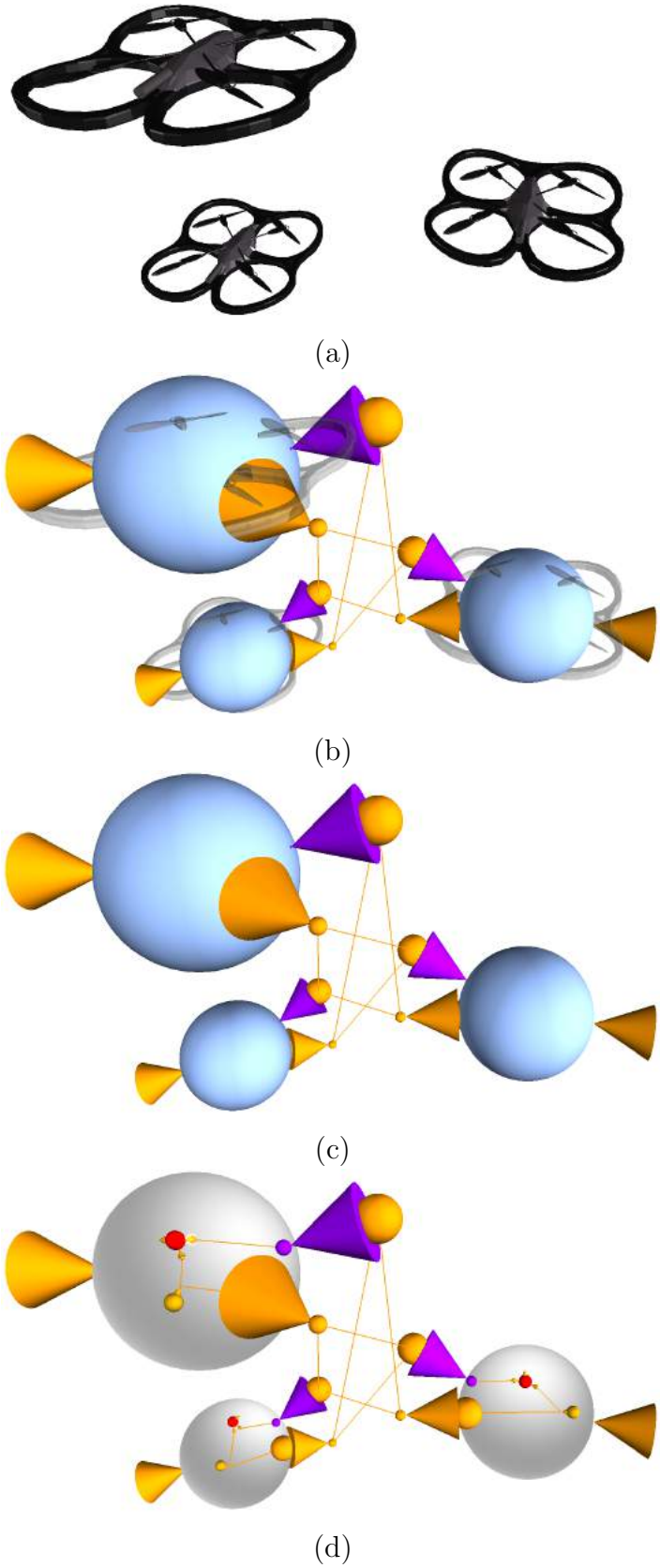


Figure 20.3: (a) Three quad-copter agents situated closely together. (b) Projection of the agents' behavioural operators and their interrelations into the agent space. (c) Focus on the behavioural network. (d) Introspection of the agents' behavioural modules reveals hierarchically nested, lower-level operators and their connectivity.

abstraction's lifespan. Confidence metrics that are too generous, i.e., that cause too long abstraction lifespans, diminish the accuracy of a simulation. Abstracted behaviours are repeatedly checked for validity by either exposing the subsumed agents to the environment and observing their behaviour again or by checking the predicates that have been identified in the abstraction process. This check can occur at fixed time intervals, at the designated end of the meta-agent's lifespan, or based on heuristics such as the degree of activity in its local environment. If the abstraction proves valid, confidence rises and the checks become less frequent. However, if the abstraction proves invalid, confidence sinks and the abstraction is either checked more often or abandoned completely.

In case of miscalculations, the simulation could be reset to a previous simulation state, adjusted and partially recomputed. This additional overhead might make it hard to reach a gain in efficiency. On the other hand, if confidence is assigned too cautiously to motifs, abstraction hierarchies do not get a chance to form in the first place. Thus, a careful balance has to be found. Learning methods as introduced in Section 20.5.6 can help find suitable parameters for concrete scenarios.

20.5.5 Execution Model

Our stated goal is to create a learning abstraction mechanism that makes as few assumptions as possible about the agents it is working with. However, in order for behavioural abstraction and hierarchical abstraction to work, the underlying execution model has to fulfil some requirements.

As mentioned before, *behavioural abstraction* requires that some of the internal rules according to which an agent operates can be suspended by an external entity. This is a natural assumption if agents observe themselves or if they can issue the rule's temporary removal (e.g., to a global simulation engine). However, if the agents are fully opaque and abstraction is performed by specialised observers, they need to be able to influence them directly. As the system designer usually has complete control over the simulation environment, it should be possible to implement such a feature within the environment directly.

For *hierarchical abstraction*, we assume that execution of the agents follows the hierarchy as well. First, root nodes are considered for execution. Their children are considered recursively, only if they are active, i.e. if they are not suspended. Deactivating child nodes instead of

removing them from the simulation entirely is necessary in order to check the abstractions' validity. Their (inactive) maintenance as part of the simulation hierarchy also serves to update their states as part of abstracted high-level behaviours.

Since simulations are usually closed systems, it is safe to assume that a benevolence assumption holds. This means that no agent in the system has an incentive to deceive the observers and information about states and state changes is provided freely and without inhibition.

20.5.6 Learning SOMO: Parameters, Knowledge Propagation, and Procreation

As an unsupervised learning approach, the self-organised middle-out learner will have to be able to learn about itself and thus become self-aware in a sense. A simple example is the requirement to learn which abstractions worked in the past and which failed to show the desired benefits. If abstractions had to be quickly dissolved, the SOMO observer that created them obviously did something wrong. Either its observations were faulty or the parameters were sub-optimal, e.g., the confidence value that is used to estimate when it is safe to assume that an interaction actually occurs repeatedly.

On the other hand, multiple SOMO observers deployed in the system should be able to learn from each other. An abstraction that has proven valid for one observer should not have to be learned by other observers in the system. Instead, patterns should be propagated and the knowledge acquired should be spread throughout the system. This way, the SOMO learner becomes an organic, learning, improving system within the system that constantly revises and improves its knowledge about the environment and itself by the meta-interaction of the individual observers.

Thus, SOMO agents learn on two levels: they adapt and improve their individual learning and abstraction parameters to become well suited for the niche they occupy in the simulation; and they exchange knowledge with each other and incorporate this knowledge in their decision making process.

The former kind of learning can be performed based on the data the agents collect and based on the perceived results of the actions performed by the agents. If a behavioural abstraction

has proven unstable, the agent can, e.g., increase the confidence value at which it abstracts behaviour. It would thus have to be more certain that a behaviour occurs repeatedly in the same fashion before abstracting it. More excitingly, however, an additional feedback loop can be added to a SOMO learner that uses the data collected by the agent to simulate different sets of parameters and the results they would have yielded. Such a simulation-within-the-simulation can use an evolutionary algorithm (EA) to evolve and test a population of parameter sets, simulate the learner's behaviour and use a fitness function that checks whether the parameters would have found abstractions that have actually proven valid. A parameter set with a high number of valid abstractions gets a higher fitness value and may be adopted. The EA can run concurrently and change the parameter settings whenever better results are obtained than possible with the current parameters. A similar approach has, e.g., been used to create and simulate new traffic light switching rules in a traffic-control scenario [657].

The latter kind of learning, in which patterns, motifs, parameter sets, etc. are propagated in the system can be implemented using gossiping algorithms [658]. These consensus approaches are built around local communication in which information is primarily exchanged with neighbours, aggregated, and spread through the system. As the communication is limited to a small number of agents, the system is scalable and since information is always disseminated along several trajectories, the system is robust. A major concern in the design of such algorithms is "eventual consensus", i.e., ensuring that at one point, all agents have access to the information. Fortunately, the SOMO learning approach does not have this requirement as even local knowledge exchange can improve its efficiency and thus, relatively simple gossiping protocols can be used.

Whenever a SOMO agents learns a new set of parameters, a new motif, or that a certain abstraction has proven valid, it can provide this information to other agents in its neighbourhood. These recipients can elect to use this information, e.g., because they are situated in a similar environment, or discard them. They can also elect to augment or redact the information and send them on to their own neighbours. This way, knowledge spreads through the system and allows the learning agents to profit from the experiences of others. Similar techniques have, e.g., been used to spread reputation information in multi-agent systems [280].

For the transmittal of information between SOMO learners, a language for the knowledge of the agents has to be defined. Apart from using it in the exchange of information, it can also

be used to store the knowledge between simulation runs. This way, different runs of the same simulation can profit from knowledge learned previously and—if the simulations are similar enough—different simulations can re-use knowledge learned previously. A SOMO learner that is repeatedly used in the same setting can thus evolve along with the simulation and improve over time.

In settings in which the simulation is highly dynamic, an additional meta-learning approach can be used. At the start of the simulation, SOMO learners are spread evenly within the simulation space. If a SOMO agent finds itself in a highly dynamic environment, with many entities to observe and many interactions, it can procreate by spawning a duplicate of itself. This new agent carries the same knowledge as its father and can become active in the same area. Thus, the SOMO system self-organises towards a structure in which learning takes place in those locations where it is most beneficial and where most interactions occur.

While the outlined meta-learning approaches should improve SOMO's ability to find valid abstractions and simplify the simulations, they incur additional computational cost as well as increase the memory requirements. Therefore, the use of these faculties has to be evaluated carefully for each new simulation setting and the trade-off between the resources required for meta-learning and the benefit has to be analysed.

20.6 Current implementations

In several publications, Sarraf Shirazi et. al present the exploration and extension of SOMO implementations in the context of biological simulations [9, 247, 605, 516, 540, 628, 515]. Therein, the application domain slightly shifted from protein-interaction networks (in context of the MAPK signalling pathway) towards cell-cell/cell-membrane interaction systems (in context of blood coagulation processes). More importantly, the model representations underwent an evolution as well: Sarraf Shirazi and his colleagues (one of them is an author of this chapter, S. von Mammen) first learned clusters of intertwined functions of gene expression rates by correlating their results—initially by means of artificial neural networks, then by means of genetic algorithms. The second iteration of implementations featured rule-based multi-agent representations and sets of learning observer agents that logged and subsumed the activities of the agents in the simulation. For instance, blood platelets and fibrinogens that are stuck together

are subsumed by meta-agents with reduced rule sets and which represent the blood clot.

Current SOMO implementations have shown the effectiveness of the concept. In early experiments the number of tests performed as part of the simulation was successfully reduced. In later experiments, Sarraf Shirazi et al. were able to show that the overall performance, also considering the computational overhead needed for observing and dynamic learning, can be improved.

The original SOMO concept foresees the possibility to expose SOMO agents without prior knowledge to an arbitrary multi-agent simulation to automatically infer hierarchies of patterns from the observed processes. In order to reach this desirable goal, numerous challenges still have to be addressed. The universal deployment of the SOMO concept requires, for instance, a generic learning mechanism for identifying arbitrary patterns (e.g., learning classifier systems [452]), a universal approach to measuring and comparing confidence values and an accordingly tuned reinforcement learning mechanism, as well as a comprehensive formalisation of representation and algorithms.

An example of a meso-level abstraction algorithm with a more technical focus has been presented by Steghöfer et al. [659] with the HiSPADA algorithm. The Hierarchical Set Partitioning Algorithm for Distributed Agents forms abstraction hierarchies within an agent society based on scalability metrics. If an agent system solves a computationally intensive problem that is defined by the individual agents (such as scheduling in power management scenarios) and that can be hierarchically decomposed, intermediaries can be introduced to solve parts of the original problem. Each intermediary solves a sub-problem that is defined by the agent it directly controls. The runtime of the problem solver depends on the number of agents controlled by an intermediary. If it exceeds a certain threshold, an additional layer of intermediaries can be introduced to divide the controlled agents. An intermediary acts as a black box to the outside, much like the meta-agents in the hierarchical abstraction. However, the intermediary is not the result of a learning process based on the interaction patterns of the agents but merely a result of an internal constraint violation. Nevertheless, the concept has proven to improve scalability in large systems and provides a starting point for future research.

20.7 Awareness beyond virtuality

It has already been shown that current implementations of SOMO are capable of pruning computational complexity in multi-agent based simulations and identifying emergent processes. A broadly deployable, unbiased SOMO implementation would make it possible to compute models with large numbers of approximate constants, such as in our perceived reality. This would make it possible to integrate vast quantities of scientific facts, across all levels of scale and scientific disciplines, for consideration in simulations.

20.7.1 Integration & emergence

The result would be *virtually* unlimited computing power for models with large numbers of approximate constants—as in our perceived reality. Vast amounts of scientific facts, across all levels of scale and scientific disciplines, could be integrated for consideration in simulations. The development of an organism could be computed bottom-up from a single fertilised cell. As we believe SOMO to be principally capable of developing awareness for previously unknown emergent phenomena—both *in-silico* and *in-vivo*—the organism's systems would be identified automatically. The recognised patterns are expressed in algorithmic rather than traditional mathematic representations, and therefore human-readable and comparable to human reasoning.

20.7.2 Model inference

What is more, the SOMO concept need not be limited to virtual simulations. Heuristic learning methods could supply feasible solutions for gaps in theories, for which empiric researchers haven't provided answers yet. However, instead of limiting SOMO to virtual simulations, it could operate on top of a smart sensory network (SOMO net), an advanced wireless sensor networks (cf., e.g., [660]). Enhancing SOMO sensory nodes with effectors would further introduce the capability of self-directed inquiry. At this point, the SOMO net could turn into a self-reflective machinery similar to the one developed by Lipson and Pollack [661] that also grew, the other way round, into a system to automatically infer complex, non-linear mathemat-

ical laws from data sets by avoiding trivial invariants [662]. SOMO net enhanced in this way would be able to autonomously perform observational analysis and pro-active investigations to further accelerate the generation of comprehensive and accurate scientific models.

20.7.3 SOMO net

In addition to the sensory functionalities present in a subset of nodes of the envisioned SOMO net, all the nodes would have to provide a runtime environment for a SOMO agent. To begin with, the initialised, *networked* SOMO agent—a conceptual descendant of the SOMO observer as deployed in virtual simulation environments—would sense and transmit data to its neighbours and, in turn, aggregate any received information. The analogies to distributed learning approaches are obvious, especially in the context of wireless sensor networks [663]. However despite the common notion of a global learning task, distributed data sources, and efforts to fuse the aggregated data, the SOMO reaches further.

Quickly, a SOMO agent would learn patterns in the sensed and received, transmitted data and refine its sensing configuration and communication connectivity based on the greatest information gain: it would direct its inquiries to areas of interest, i.e., sensor ranges or nodes that provide (from its perspective) unpredictable information. Depending on the confidence values associated with the learned patterns, the original data sources would be queried once in a while in order to test the abstractions' validity.

As the learned patterns would reference the learning context, i.e., the network location and connectivity of the learning agent, the abstracted information can be passed down the network, enriching the other agents' data bases, without causing confusion. Whenever possible, patterns could be subsumed in higher level abstractions, the validation process stretching across the network.

The self-organised, decentralised learning and validation algorithm would ensure that the system under observation is described at several levels of abstraction, based on the input on numerous nodes with their individual perspectives. At the same time, it would ensure that the processing and communication costs of the networked nodes is minimised—which is of crucial importance for the efficacy and longevity of a wireless sensor network.

20.7.4 SOMO after me

SOMO and SOMO nets would make correlations between processes apparent that have never been thought of before. These new insights, could, due to the immense complexity that SOMO promises to handle, help to build sustainable, progressive, evolving economic and ecological infrastructures for the great challenges of human kind.

At the same time, accessible methodologies for large-scale data modelling and exploration would become an (even more) important limiting factor. In order to counter this arising challenge, we have been developing INTO3D, an integrated visual programming and simulation environment [653]. Combined with SOMO's computing abilities such environments could make model-building and simulation feasible and attractive to non-scientists, or rather, they could turn anyone into a scientist and revolutionise everyday life.

20.8 The future of SOMO

In summary, the SOMO algorithm and SOMO nets hold the promise of revealing hitherto unsuspected correlations between processes. Such new insights, and the immense complexity that SOMO can handle, would help to build the sustainable, progressive and evolving economic and ecological infrastructures for tackling the major challenges humankind faces today. Our current work on SOMO is focused on pattern detection in observed interactions and the possibilities for propagating knowledge about abstractions through the system. Once the implementations of SOMO have reached maturity, we envisage that research can shift to analysing how the learned abstractions and features correlate with the behaviours we find in higher order emergent phenomena. Whether we will find striking similarities, or instead discover these to be two completely different forms of complex systems, remains an exciting open question at this time.

Chapter 21

Conclusion

Interactive self-organisation represents the next step of evolution in self-organisation research. By introducing the user into the simulation-loop, he can immensely benefit from the concept of self-organisation. After all, self-organisation not only describes a complex system with a potential for emergent phenomena, but it also shows the user how local and global processes are linked, it gives the user the opportunity to change them bottom-up, top-down or middle-out. Novel computer scientific techniques make this happen, both in terms of algorithmic, technical solutions and approaches of user interaction and visualisation. In this habilitation thesis, we cumulated eighteen published scientific works that introduced according techniques and application scenarios. In order to fully harness the potential of interactive self-organisation, research towards all of the aspects touched upon in this thesis as well as their integration and unification has to continue. As a result, one of the primary goals of interactive self-organisation, next to its application, needs to be a consolidated framework that seamlessly combines management, visual programming, exploration, inspection, modulation, and analysis of vast, multi-scale self-organising system models and simulations. Work on the following research tasks would contribute to this goal.

In order to make inter-agent relationships and agent behaviours editable through graph-based visual interfaces, it would help to find ways to establish different meanings for different edges. In Chapter 12 labels fulfil this requirement, but there are myriad ways of visualising directed edges and it would be rather beneficial to utilise this freedom of representation in order to convey meaning. One could, for instance, change the basic graphical properties of the edges, make

them slimmer or wider, scale their heads, change their colours, etc. depending on the concrete meaning of a relationship or merely on its general classification. While all these options seem to be equally valid, some of them work better than others, which results in a clear mandate for visualisation research. It also presupposes the design of a consistent vocabulary that emerges from a versatile, axiomatic, extensible representation. Additional questions that arise in the context of local rule definition relate to the deeper integration of graphical and textual editing, i.e. whether textual input should be omitted, how it could be integrated in a non-disruptive way, and how provision of detailed information can be warranted in an otherwise visual modulation environment.

Also, at this point, we only have little insight into the optimal work flow of modelling behavioural logic and model properties (other than linear transformations) in simulation spaces. It is an obvious choice to support this endeavour with the latest virtual reality gear not only to increase the accuracy of spatial interactions but to maximise the benefit of three-dimensionally rendered spatial relationships, too. There are all sorts of design decisions that need to be inquired about, such as the relationship between 3D user interface elements and heads-up display information, the utilisation of drag-and-drop techniques, or the seamless integration of navigation. Furthermore, we need to better understand the benefits and shortcomings of isolated editing spaces and re-design the immersive 3D modelling interfaces accordingly. Only a systematic taxonomy to categorise and orderly draw out the interaction tasks involved in modulation of self-organising systems can help to identify all the intricacies of the process and to unify them. Generally speaking, human-computer interaction research in terms of navigation, selection, manipulation and control of self-organising systems has just recently begun, while emerging virtual reality technologies promise to revolutionise the interaction possibilities with spatial systems.

Despite all efforts, users of visual programming environments make mistakes. Due to the prevention of syntactical errors by means of the user interface, they are mostly of semantical nature. Established techniques such as the utilisation of puzzle-like visual environments, that allow the user to combine programmatic building blocks, need to be adjusted and refined, if translated into the modulation spaces of interactive self-organisation. Programming in three dimensions also opens up new paths towards dynamic and static data typing. The desired data types could be manually built in graph-based editors and scaled replica of their visual

representations could be utilised as connectors for the respective edges. This would allow to recognise not only a mismatch but even indicate the part of a data structure where it occurs. When debugging a model, the amalgamation of modelling space and simulation space yields the great advantage that, for example, wrong assignments or poorly configured interactions can be immediately discovered—if the modeller has a clear notion about the system and it is visually represented, deviations from his cognitive model can be identified faster.

The specification of high-level goals is supposed to complement the process of programming low-level details. This functionality is especially important in the context of interactive self-organisation, since the low-level behaviours of a target system might not be known to the modeller (in advance). Typically, high-level goals are specified as mathematical constraints, logic terms, or algorithmically. An according, comprehensive visual vocabulary does not exist, yet. Chapter 17 details the possibility to provide graphical input constraints, whether they represent actual spatial constraints in 2D or 3D, or whether the user can predefine time series. Additional research needs to be conducted to innovate effective interfaces to edit these constraints, to specify their boundaries and data types, to scale them, and, ideally, to relate them to their target entities, interactions, processes and structures. Whenever possible, widgets should be made available to blend these activities into a general interactive self-organisation framework.

Scaling capacities of computation as well as storage and retrieval is of great importance in interactive self-organisation, too. The user expects from any sort of content creation tool that even his most minute changes can be recovered, if a power failure occurs, that he can go back arbitrary steps in his interaction history, that he can branch off simulation paths to explore alternative model configurations, that he can review individual experiments, compare them, work on them collaboratively, possibly remotely, etc. Creating such functionalities is disentangled by following the component software engineering pattern (Chapter 15). Yet, depending on the model sizes, these operations can be rather expensive and their efficient execution is tied to an efficient hardware setup. Given a specific networked infrastructure, the problem is to optimally manage data persistence alongside the distributed organisation of multiple highly optimised simulation engines. To this end, we promote research into a unified kernel model for distributed component engines and distributed scalable database systems.

It is mandatory to introduce new components to interactive self-organisation frameworks, e.g. for networking, and improving the underlying engines, e.g. graphics engines making use of

state-of-the-art shader hardware. We have, for example, identified the integration of the latest augmented reality rendering techniques as a greatly beneficial asset. Augmented reality is an important outlet for interactive self-organisation as it directly translates calculated results into real contexts. In our experiments [112], we were able to augment reality with interactive self-organising models of so-called biohybrid systems. Although the testers were able to achieve the set goals, the lack of integration into the real lighting settings had a consistently damaging impact on both usability and user experience. Hardware limitations (especially the fact of wearing tethered gear) posted the other main issues in our experiments. Gladly, these hurdles are being taken care of by the technology industry. Another example of an important extension of component engines is the integration of the FLEX particle-based physics engine into our modulation system for developmental biology. It allows us to outsource costly calculations to the graphics processing unit and, more importantly, empowers us to model the physical interactions of material with different properties in a shared simulation environment. These include rigid bodies, deformable bodies and fluids. As pointed out in Chapter 5, it has been shown that in simulations of developmental biology, the consideration of physically accurate models plays an important role to retrace actual self-organising biological processes.

With respect to runtime optimisation, the promising work around self-organised middle-out abstraction should be continued. In order to foster basic research in this field, we suggest collective motion as the targeted model domain. It is well-studied, it covers a wide range of phenomena, including laminar and turbulent flows, and various flock and group formations, and it is limited to reactive agents. Moreover, we propose the deployment of Organic Computing techniques in order to realise the original SOMO concept from 2011. The steps towards a versatile, robust SOMO implementation include retracing data sets of established domain models, the definition and implementation of error measures for comparing simulated processes, and the deployment of different learning mechanisms used by observer/controller agents that can be immersed in the targeted model simulation. An extensive investigation of the resulting performances regarding the given models with varying configurations as well as with respect to hierarchical model abstraction methods needs to follow suit. To complete the research and development cycle, one would have to redesign and hone the application programming interface of OSOM to make it broadly applicable. Another direction for SOMO research could investigate the possibilities to run the overhead of OSOM in parallel to the domain model simulation or to outsource it to the GPU. As outlined in Part I, building on SOMO one could make model optimisation tangible,

allow the users to help in the abstraction process, to guide the self-organisation dissemination and evolution of abstraction agents and their knowledge bases.

Interactive self-organisation research can hardly come to fruition, if it is not linked to specific use cases. The demonstration of the merits of applied interactive self-organisation not only gives the opportunity to analyse the feasibility and track the impact of a novel approach but it also plays an important role in guiding the direction of interactive self-organisation research. The methods and results presented in this thesis are mainly independent of technical self-organising solutions. However, the current trends in augmented reality and robotics allow us to acknowledge that technical self-organising solutions will occur in all the presented application domains, and potentially in many more, in the near future. This perspective spurs the motivation to conduct interactive self-organisation research all the more, as its demand will soar alongside its emerging applications and successes.

Index

Bibliography

[1] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[2] G. Theraulaz and E. Bonabeau, "Modelling the collective building of complex architectures in social insects with lattice swarms," *Journal of Theoretical Biology*, vol. 177, no. 4, pp. 381–400, 1995.

[3] S. von Mammen and C. Jacob, "The Evolution of Swarm Grammars: Growing Trees, Crafting Art and Bottom-Up Design," *IEEE Computational Intelligence Magazine*, Aug. 2009.

[4] S. von Mammen, S. Novakowski, G. Hushlak, and C. Jacob, "Evolutionary swarm design: How can swarm-based systems help to generate and evaluate designs?," *DESIGN Principles & Practices: An International Journal*, vol. 3, pp. 371–386, 2009.

[5] M. West, "Evolve your hierarchy." Published online `http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/`, January 2007.

[6] S. Havre, B. Hetzler, and L. Nowell, "Themerivertm: In search of trends, patterns, and relationships," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 9–20, 2002.

[7] C.-Y. Huang and J. E. Ferrell, "Ultrasensitivity in the mitogen-activated protein kinase cascade," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 93, pp. 10078–10083, Sept. 1996.

[8] B. N. Kholodenko, "Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades," *European journal of biochemistry / FEBS*, vol. 267, pp. 1583–1588, Mar. 2000.

[9] A. Sarraf Shirazi, S. von Mammen, and C. Jacob, "Adaptive modularization of the mapk signaling pathway using the multiagent paradigm," in *Parallel Problem Solving from Nature – PPSN XI*, vol. 6239 of *Lecture Notes in Computer Science*, pp. 401–410, Krakow, Poland: Springer Verlag, 2010.

[10] S. Rasmussen, N. A. Baas, B. Mayer, M. Nilsson, and M. W. Olesen, "Ansatz for dynamical hierarchies," *Artificial Life*, vol. 7, pp. 329–353, Mar. 2002.

[11] S. Marras, S. S. Killen, J. Lindström, D. J. McKenzie, J. F. Steffensen, and P. Domenici, "Fish swimming in schools save energy regardless of their spatial position," *Behavioral Ecology and Sociobiology*, vol. 69, no. 2, pp. 219–226, 2015.

[12] T. J. Czaczkes, C. Grüter, and F. L. Ratnieks, "Trail pheromones: An integrative view of their role in social insect colony organization," *Annual review of entomology*, vol. 60, pp. 581–599, 2015.

[13] T. Parmentier, W. Dekoninck, and T. Wenseleers, "Context-dependent specialization in colony defence in the red wood ant formica rufa," *Animal Behaviour*, vol. 103, pp. 161–167, 2015.

[14] D. Fouquet, A. Costa-Leonardo, R. Fournier, S. Blanco, and C. Jost, "Coordination of construction behavior in the termite procornitermes araujoi: structure is a stronger stimulus than volatile marking," *Insectes sociaux*, vol. 61, no. 3, pp. 253–264, 2014.

[15] P. S. Andrews, F. A. Polack, A. T. Sampson, S. Stepney, and J. Timmis, "The cosmos process version 0.1: A process for the modelling and simulation of complex systems," *Department of Computer Science, University of York, Tech. Rep. YCS-2010-453*, 2010.

[16] F. A. C. Polack, "Proposals for validation of simulations in science," in *Proceedings of the 2010 Workshop on Complex Systems Modelling and Simulation*, (Odense, Denmark), pp. pp. 51–74, Luniver Press, 2010.

[17] J.-L. Giavitto and O. Michel, "Data structure as topological spaces," *Unconventional Models of Computation*, pp. 137–150, 2002.

[18] J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz, *Modelling and Simulation of biological processes in the context of genomics*, ch. Computational Models for Integrative and Developmental Biology, pp. 12–17. Hermes, July 2002.

[19] J.-L. Giavitto, "Topological collections, transformations and their application to the modeling and the simulation of dynamical systems," in *Rewriting Techniques and Applications*, pp. 208–233, Springer, 2003.

[20] J. D. Foley, V. L. Wallace, and P. Chan, "The human factors of computer graphics interaction techniques," *Computer Graphics and Applications, IEEE*, vol. 4, no. 11, pp. 13–48, 1984.

[21] B. Preim and R. Dachselt, *Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces*. Springer-Verlag, 2015.

[22] M. Weiser, "Ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[23] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[24] M. Sitti, H. Ceylan, W. Hu, J. Giltinan, M. Turan, S. Yim, and E. Diller, "Biomedical applications of untethered mobile milli/microrobots," *Proceedings of the IEEE*, vol. 103, no. 2, pp. 205–224, 2015.

[25] P. C. Salmon and P. L. Meissner, "Mobile bot swarms: They're closer than you might think!," *Consumer Electronics Magazine, IEEE*, vol. 4, no. 1, pp. 58–65, 2015.

[26] P. Beesley, S. Hirosue, J. Ruxton, M. Trankle, and C. Turner, *Responsive Architectures: Subtle Technologies*. Riverside Architectural Press, 2006.

[27] J. Banks *et al.*, *Handbook of simulation*. Wiley Online Library, 1998.

[28] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in artificial life*, pp. 704–720, Springer, 1995.

[29] S. Tomforde, *Runtime adaptation of technical systems: An architectural framework for self-configuration and self-improvement at runtime*. Südwestdeutscher Verlag für Hochschulschriften, 2012. ISBN: 978-3838131337.

[30] B. Williams, *Microsoft Flight Simulator as a training aid: a guide for pilots, instructors, and virtual aviators*. Aviation Supplies & Academics, 2006.

[31] P. Backlund, H. Engström, M. Johannesson, and M. Lebram, "Games for traffic education: An experimental study of a game-based driving simulator," *Simulation & Gaming*, 2008.

[32] R. Harper, *Inside the smart home*. Springer Science & Business Media, 2003.

[33] H. Lipson and M. Kurman, *Fabricated: The new world of 3D printing*. John Wiley & Sons, 2013.

[34] E. J. Topol, "Individualized medicine from prewomb to tomb," *Cell*, vol. 157, no. 1, pp. 241–253, 2014.

[35] T. A. Nguyen and M. Aiello, "Energy intelligent buildings based on user activity: A survey," *Energy and buildings*, vol. 56, pp. 244–257, 2013.

[36] F. Macedonio, E. Drioli, A. Gusev, A. Bardow, R. Semiat, and M. Kurihara, "Efficient technologies for worldwide clean water supply," *Chemical Engineering and Processing: Process Intensification*, vol. 51, pp. 2–17, 2012.

[37] B. Sedacca, "Hand built by lasers [additive layer manufacturing]," *Engineering & Technology*, vol. 6, no. 1, pp. 58–60, 2011.

[38] S. Wolfram, *The Mathematica Book, Fifth Edition*. Wolfram Media Inc., 2003.

[39] D. Redfern and C. Campbell, *The MATLAB® 5 Handbook*. Springer Science & Business Media, 2012.

[40] J. W. Eaton, *GNU Octave: A high-level interactive language for numerical computations, 3rd ed.* e Free Software Foundation, Inc., Boston, USA, 2006.

[41] The Sage Development Team, *Sage Tutorial, Release 6.9*. The Sage Development Team, 2015.

[42] Computational Algebra Group, *The Magma Handbook, V2.21*. Computational Algebra Group, University of Sidney, 2015.

[43] D. Redfern, *The maple handbook: maple V release 4*. Springer Science & Business Media, 2012.

[44] O. Dimigen and U. Reinacher, "Active vision plugin: An open-source matlab tool for saccade-and fixation-related eeg analysis," in *Perception*, vol. 41, (London, UK), pp. 246–246, Pion Ltd., 2012.

[45] J. Mikulka, "Matlab extension for 3dslicer: A robust mr image processing tool," in *Progress in Electromagnetics Research Symposium, Proceedings*, (Guangzhou, China), pp. 1875–1860, 2014.

[46] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013.

[47] L. Tesfatsion, *Handbook of Computational Economics*, vol. 2, ch. Agent-Based Computational Economics: A Constructive Approach to Economic Theory, pp. 831–880. Elsevier, 2006.

[48] J. M. Epstein and R. Axtell, *Growing artificial societies: social science from the bottom up*. Brookings Institution Press, 1996.

[49] S. C. Brenner and C. Carstensen, "Finite element methods," *Encyclopedia of computational mechanics*, 2004.

[50] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 154–159, Eurographics Association, 2003.

[51] S. M. Hosseini and J. J. Feng, "A particle-based model for the transport of erythrocytes in capillaries," *Chemical Engineering Science*, vol. 64, no. 22, pp. 4488–4497, 2009.

[52] P. Li, P. Mirchandani, and X. Zhou, "Metrosim: A hierarchical multi-resolution traffic simulator for metropolitan areas: Architecture, challenges and solutions," in *Transportation Research Board 94th Annual Meeting*, pp. number 15–3201, 2015.

[53] M. Wooldridge, *An introduction to multiagent systems*. Wiley. com, 2008.

[54] J. Denzinger and M. Kordt, "Evolutionary on-line learning of cooperative behavior with situation-action-pairs," in *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000)*, (Boston, MA, USA), pp. 103–110, 2000.

[55] J. Denzinger and C. Winder, "Combining coaching and learning to create cooperative character behavior," in *Proceeding of the Symposium on Computational Intelligence and Games*, (Essex University, Colchester, Essex), IEEE, April 2005.

[56] M. Niazi and A. Hussain, "Agent-based computing from multi-agent systems to agent-based models: a visual survey," *Scientometrics*, vol. 89, no. 2, pp. 479–499, 2011.

[57] S. Tisue and U. Wilensky, "Netlogo: Design and implementation of a multi-agent modeling environment," in *Agent 2004: Conference on Social Dynamics*, (Chicago, IL), 2004.

[58] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with repast simphony," *Complex adaptive systems modeling*, vol. 1, no. 1, pp. 1–26, 2013.

[59] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan, "Mason: A new multi-agent simulation toolkit," in *Proceedings of the 2004 swarmfest workshop*, vol. 8, p. 44, 2004.

[60] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The swarm simulation system: A toolkit for building multi-agent simulations," tech. rep., Santa Fe Institute, Santa Fe, New Mexico, USA, 1996.

[61] S. F. Railsback, S. L. Lytinen, and S. K. Jackson, "Agent-based simulation platforms: Review and development recommendations," *Simulation*, vol. 82, no. 9, pp. 609–623, 2006.

[62] C. Nikolai and G. Madey, "Tools of the trade: A survey of various agent based modeling platforms," *Journal of Artificial Societies and Social Simulation*, vol. 12, 2009.

[63] P. S. Paul, S. Goon, and A. Bhattacharya, "History and comparative study of modern game engines," *International Journal of Advanced Computer and Mathematical Sciences*, vol. 3, no. 2, pp. 245–249, 2012.

[64] J. Haas, *A History of the Unity Game Engine*. PhD thesis, Worcester Polytechnic Institute, 2014.

[65] M. M. Jones, "On-line simulation," in *Proceedings of the 1967 22Nd National Conference*, ACM '67, (New York, NY, USA), pp. 591–599, ACM, 1967.

[66] D. R. Kalasky and D. A. Davis, "Computer animation with cinema," in *Proceedings of the 23rd conference on Winter simulation*, pp. 122–127, IEEE Computer Society, 1991.

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

[68] P. C. Bell and R. M. O'keefe, "Visual interactive simulation—history, recent developments, and major issues," *Simulation*, vol. 49, no. 3, pp. 109–116, 1987.

[69] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[70] J. Klein, "breve: a 3d simulation environment for multi-agent simulations and artificial life.." Published online `http://www.spiderland.org/`, October 2008.

[71] 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*, pp. 151–182, 2009.

[72] G. Nicolis and C. Rouvas-Nicolis, "Complex systems," *Scholarpedia*, vol. 2, no. 11, p. 1473, 2007.

[73] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, pp. 81–85, Apr. 2009.

[74] N. Magnenat-Thalmann, O. Ratib, and H. F. Choi, *3D Multiscale Physiological Human*. Springer, 2014.

[75] D. P. Nickerson, D. Ladd, J. R. Hussan, S. Safaei, V. Suresh, P. J. Hunter, and C. P. Bradley, "Using cellml with opencmiss to simulate multi-scale physiology," *Frontiers in bioengineering and biotechnology*, vol. 2, 2014.

[76] R. M. M. Vaquero, J. Rzepecki, K.-I. Friese, and F.-E. Wolter, "Visualization and user interaction methods for multiscale biomedical data," in *3D Multiscale Physiological Human*, pp. 107–133, Springer, 2014.

[77] J. Cebulla, E. Kim, K. Rhie, J. Zhang, and A. P. Pathak, "Multiscale and multi-modality visualization of angiogenesis in a human breast cancer model," *Angiogenesis*, vol. 17, no. 3, pp. 695–709, 2014.

[78] M. Turk, "Multimodal interaction: A review," *Pattern Recognition Letters*, vol. 36, pp. 189–195, 2014.

[79] M. Dunleavy and C. Dede, "Augmented reality teaching and learning," in *Handbook of research on educational communications and technology*, pp. 735–745, Springer, 2014.

[80] R. T. Azuma *et al.*, "A survey of augmented reality," *Presence*, vol. 6, no. 4, pp. 355–385, 1997.

[81] J. M. Slack, *Essential developmental biology*. John Wiley & Sons, 2009.

[82] S. F. Gilbert, *Developmental Biology*. Sinauer Associates, Inc, 10th ed., 2013.

[83] I. Salazar-Ciudad, J. Jernvall, and S. Newman, "Mechanisms of pattern formation in development and evolution," *Development*, vol. 130, no. 10, pp. 2027–2037, 2003.

[84] D. Duboule, *Guidebook to the homeobox genes*. Oxford Univ. Press, New York, NY (United States), 1995.

[85] M. Théry and M. Bornens, "Cell shape and cell division," *Current opinion in cell biology*, vol. 18, no. 6, pp. 648–657, 2006.

[86] Q. Xu, H. Jamniczky, D. Hu, R. M. Green, R. S. Marcucio, B. Hallgrimsson, and W. Mio, "Correlations between the morphology of sonic hedgehog expression domains and embryonic craniofacial shape," *Evolutionary Biology*, pp. 1–8, 2015.

[87] T. Eissing, L. Kuepfer, C. Becker, M. Block, K. Coboeken, T. Gaub, L. Goerlitz, J. Jaeger, R. Loosen, B. Ludewig, M. Meyer, C. Niederalt, M. Sevestre, H.-U. Siegmund, J. Solodenko, K. Thelen, U. Telle, W. Weiss, T. Wendl, S. Willmann, and J. Lippert, "A computational systems biology software platform for multiscale modeling and simulation: integrating whole-body physiology, disease biology, and molecular reaction networks," *Frontiers in Physiology*, vol. 2, February 2011.

[88] 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*, vol. 41, pp. 429–452, 2012.

[89] D. Noble, *The music of life*. Oxford University Press, 2006.

[90] M. P. Mota, I. T. Monteiro, J. J. Ferreira, C. Slaviero, and C. S. de Souza, "On signifying the complexity of inter-agent relations in agentsheets games and simulations," in *Proceedings of the 31st ACM international conference on Design of communication*, pp. 133–142, ACM, 2013.

[91] T. Davison and J. Denzinger, "The huddle: Combining ai techniques to coordinate a player's game characters," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pp. 203–210, IEEE, 2012.

[92] 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*, pp. 173 –176, sept. 2006.

[93] L. Spector, J. Klein, C. Perry, and M. Feinstein, "Emergence of collective behavior in evolving populations of flying agents," *Genetic Programming and Evolvable Machines*, vol. 6, no. 1, pp. 111–125, 2005.

[94] T. Vicsek and A. Zafeiris, "Collective motion," *Physics Reports*, vol. 517, no. 3, pp. 71–140, 2012.

[95] I. Derényi and T. Vicsek, "Cooperative transport of Brownian particles," *J. Phys. I (France) Phys Rev Lett*, vol. 75, p. 374, 1994.

[96] J. McLurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt, "Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots.," in *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, pp. 72–75, 2006.

[97] A. M. Naghsh, J. Gancet, A. Tanoto, and C. Roast, "Analysis and design of human-robot swarm interaction in firefighting," in *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, pp. 255–260, IEEE, 2008.

[98] L. Pollini, M. Niccolini, M. Rosellini, and M. Innocenti, "Human-swarm interface for abstraction based control," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference, Chicago, IL, USA*, pp. 10–13, 2009.

[99] A. Kolling, S. Nunnally, and M. Lewis, "Towards human control of robot swarms," in *Proceedings of the seventh annual ACM/IEEE international conference on human-robot interaction*, pp. 89–96, ACM, 2012.

[100] K. Sycara, C. Lebiere, Y. Pei, D. Morrison, Y. Tang, and M. Lewis, "Abstraction of analytical models from cognitive models of human control of robotic swarms," in *Proceedings of International Conference on Cognitive Modeling (ICCM)*, (Groningen, the Netherlands), pp. 13–19, University of Groningen, 2015.

[101] A. Tarantola, *Inverse problem theory and methods for model parameter estimation*. siam, 2005.

[102] R. C. Aster, B. Borchers, and C. H. Thurber, *Parameter estimation and inverse problems*. Academic Press, 2011.

[103] J. Steele and N. Iliinsky, *Beautiful visualization*. O'Reilly Media, Inc., 2010.

[104] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "The state of the art in visualizing dynamic graphs," in *Proceedings of the Eurographics Conference on Visualization*, (Swansea, UK), IEEE, 2014.

[105] T. Ellis, J. F. Heafner, and W. Sibley, "The grail project: An experiment in man-machine communications," tech. rep., DTIC Document, 1969.

[106] B. A. Myers, "Visual programming, programming by example, and program visualization: a taxonomy," in *ACM SIGCHI Bulletin*, vol. 17, pp. 59–66, ACM, 1986.

[107] D. D. Hils, "Visual languages and computing survey: Data flow visual programming languages," *Journal of Visual Languages & Computing*, vol. 3, no. 1, pp. 69–101, 1992.

[108] J. Trower and J. Gray, "Creating new languages in blockly: Two case studies in media computation and robotics (abstract only)," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, (New York, NY, USA), pp. 677–677, ACM, 2015.

[109] M. M. Burnett, A. Goldberg, and T. G. Lewis, *Visual object-oriented programming: concepts and environments*. Manning Publications Co., 1995.

[110] W. Citrin, M. Doherty, and B. Zorn, "The design of a completely visual object-oriented programming language," *Visual Object-Oriented Programming: Concepts and Environments. Prentice-Hall, New York*, 1995.

[111] 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, (New York, NY, USA), pp. 142–143, ACM, 1993.

[112] M. Wahby, M. Divband Soorati, S. von Mammen, and H. Hamann, "Evolution of controllers for robot-plant bio-hybdrids: A simple case study using a model of plant growth and motion," in *Proceedings of 25. Workshop Computational Intelligence*, (Dortmund), pp. 67–86, KIT Scientific Publishing, November 2015.

[113] A. Spicher, O. Michel, and J.-L. Giavitto, "A topological framework for the specification and the simulation of discrete dynamical systems," *Cellular Automata*, pp. 238–247, 2004.

[114] A. Spicher, O. Michel, and J.-L. Giavitto, "Interaction-based simulations for integrative spatial systems biology," in *Understanding the Dynamics of Biological Systems* (W. Dubitzky, J. Southgate, and H. Fuß, eds.), pp. 195–231, Springer New York, 2011.

[115] C. Müller-Schloer, H. Schmeck, and T. Ungerer, eds., *Organic Computing - A Paradigm Shift for Complex Systems*. Autonomic Systems, Birkhäuser Verlag, 2011.

[116] A. Husselmann and K. Hawick, "Spatial data structures, sorting and gpu parallelism for situated-agent simulation and visualisation," tech. rep., Tech. Rep. CSTN-156, Computer Science, Massey University, 2012.

[117] S. von Mammen and C. Jacob, "The spatiality of swarms — quantitative analysis of dynamic interaction networks," in *Proceedings of Artificial Life XI*, pp. 662–669, MIT Press, 2008.

[118] L.-S. Wang, P. S. Krishnaprasad, and J. Maddocks, "Hamiltonian dynamics of a rigid body in a central gravitational field," *Celestial Mechanics and Dynamical Astronomy*, vol. 50, no. 4, pp. 349–386, 1990.

[119] M. Trenti and P. Hut, "N-body simulations (gravitational)," vol. 3, no. 5, p. 3930, 2008.

[120] M. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. of IMA Conference on Mathematics of Surfaces*, vol. 1, pp. 602–608, 1998.

[121] S. von Mammen and J.-P. Steghöfer, *The Computer after Me: Awareness and Self-Awareness in Autonomic Systems*, ch. Bring it on, Complexity! Present and future of self-organising middle-out abstraction. World Scientific Publishing, 2014.

[122] S. A. Bhuiyan, S. Novakowski, M. Paget, H. Jamniczky, and C. Jacob, "Augmented reality in the lindsay virtual human: Adding a new dimension in tablet-based medical education," *Journal of Undergraduate Research in Alberta*, vol. 2, no. 2, p. 2, 2012.

[123] V. Sarpe and C. Jacob, "Simulating the decentralized processes of the human immune system in a virtual anatomy model," *BMC Bioinformatics*, vol. 14, pp. 1–18, 2013.

[124] D. W.-K. Yuen, "Lindsay virtual nephron: From physics to physiology," *Journal of Undergraduate Research in Alberta*, vol. 2, no. 2, p. 21, 2012.

[125] A. Esmaeili, T. Davison, A. Wu, J. Alcantara, and C. Jacob, "Prokaryo: an illustrative and interactive computational model of the lactose operon in the bacterium escherichia coli," *BMC bioinformatics*, vol. 16, no. 1, p. 1, 2015.

[126] J. K. Tworek, H. A. Jamniczky, C. Jacob, B. Hallgrímsson, and B. Wright, "The lindsay virtual human project: an immersive approach to anatomy and physiology," *Anatomical sciences education*, vol. 6, no. 1, pp. 19–28, 2013.

[127] P. A. Guze, "Using technology to meet the challenges of medical education," *Transactions of the American Clinical and Climatological Association*, vol. 126, p. 260, 2015.

[128] Y.-L. Ng, V. Mann, S. Rahbaran, J. Lewsey, and K. Gulabivala, "Outcome of primary root canal treatment: systematic review of the literature–part 1. effects of study characteristics on probability of success," *International endodontic journal*, vol. 40, no. 12, pp. 921–939, 2007.

[129] J. Rose, J. Buchanan, and D. Sarrett, "The dentsim system," *J Dent Educ*, vol. 63, no. 5, pp. 421–3, 1999.

[130] P. Pohlenz, A. Gröbe, A. Petersik, N. Von Sternberg, B. Pflesser, A. Pommert, K.-H. Höhne, U. Tiede, I. Springer, and M. Heiland, "Virtual dental surgery as a new

educational tool in dental school," *Journal of Cranio-Maxillofacial Surgery*, vol. 38, no. 8, pp. 560–564, 2010.

[131] M. Heiland, N. Sternberg-Gospos, B. Pflesser, D. Schulze, K. H. Höhne, R. Schmelzle, and A. Petersik, "Virtual simulation of dental surgery using a three-dimensional computer model with a force feedback system," *Mund-, Kiefer- und Gesichtschirurgie*, vol. 8, no. 3, pp. 163–166, 2004.

[132] M. Kolesnikov, M. Zefran, A. D. Steinberg, and P. G. Bashook, "Periosim: Haptic virtual reality simulator for sensorimotor skill acquisition in dentistry," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 689–694, IEEE, 2009.

[133] L. Kim, Y. Hwang, S. H. Park, and S. Ha, "Dental training system using multi-modal interface," *Computer-Aided Design and Applications*, vol. 2, no. 5, pp. 591–598, 2005.

[134] B. Tse, W. Harwin, A. Barrow, B. Quinn, M. Cox, *et al.*, "Design and development of a haptic dental training system-haptel," in *Haptics: Generating and Perceiving Tangible Sensations*, pp. 101–108, Springer, 2010.

[135] Sensable Technologies, "Phantom omni haptic device." Published online `http://www.dentsable.com/haptic-phantom-omni.htm`, December 2014.

[136] D. Drasdo, S. Hoehme, and M. Block, "On the role of physics in the growth and pattern formation of multi-cellular systems: what can we learn from individual-cell based models?," *Journal of Statistical Physics*, vol. 128, no. 1-2, pp. 287–345, 2007.

[137] O. Hamant, M. G. Heisler, H. Jönsson, P. Krupinski, M. Uyttewaal, P. Bokov, F. Corson, P. Sahlin, A. Boudaoud, E. M. Meyerowitz, *et al.*, "Developmental patterning by mechanical signals in arabidopsis," *science*, vol. 322, no. 5908, pp. 1650–1655, 2008.

[138] M. Uyttewaal, J. Traas, and O. Hamant, "Integrating physical stress, growth, and development," *Current opinion in plant biology*, vol. 13, no. 1, pp. 46–52, 2010.

[139] J. Disset, S. Cussat-Blanc, and Y. Duthen, "Self-organization of symbiotic multicellular structures," in *the Fourteenth International Conference on the Synthesis and Simulation of Living Systems-ALIFE 2014*, pp. pp–541, 2014.

[140] Y. Bar-Cohen, ed., *Biomimetics: Biologically Inspired Technologies*. Taylor & Francis, 2006.

[141] I. Mazzoleni, *Architecture Follows Nature-Biomimetic Principles for Innovative Design*, vol. 2. CRC Press, 2013.

[142] D. W. Thompson, *On Growth and Form: An Abridged Edition*. Cambridge, UK: Cambridge University Press, 2008.

[143] MIT Media Lab, "Aesthetics and computation group." Published online `http://processing.org/`, 2010.

[144] M. Weinstock, "Self-organisation and material constructions," *Architectural Design*, vol. 76, no. 2, pp. 34–41, 2006.

[145] V. Singh and N. Gu, "Towards an integrated generative design framework," *Design Studies*, vol. 33, no. 2, pp. 185–207, 2012.

[146] E. Baharlou and A. Menges, "Generative agent-based design computation," in *eCAADe 2013: Computation and Performance–Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe, Delft, The Netherlands, September 18-20, 2013*, Faculty of Architecture, Delft University of Technology; eCAADe (Education and research in Computer Aided Architectural Design in Europe), 2013.

[147] M. Hensel, A. Menges, and M. Weinstock, *Emergent technologies and design: towards a biological paradigm for architecture*. Routledge, 2013.

[148] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[149] K. Petersen, P. Bardunias, N. Napp, J. Werfel, R. Nagpal, and S. Turner, "Arrestant property of recently manipulated soil on macrotermes michaelseni as determined through visual tracking and automatic labeling of individual termite behaviors," *Behavioural processes*, vol. 116, pp. 8–11, 2015.

[150] A. Khuong, J. Gautrais, A. Perna, C. Sbaï, M. Combe, P. Kuntz, C. Jost, and G. Theraulaz, "Stigmergic construction and topochemical information shape ant nest architec-

ture," *Proceedings of the National Academy of Sciences*, vol. 113, no. 5, pp. 1303–1308, 2016.

[151] X. Tu and D. Terzopoulos, "Artificial fishes: Physics, locomotion, perception, behavior," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 43–50, ACM, 1994.

[152] J. E. Boyd, G. Hushlak, and C. Jacob, "Swarmart: Interactive art from swarm intelligence," in *Proceedings of the 12th annual ACM international conference on Multimedia*, (New York, NY, USA), pp. 628–635, ACM Press, 2004.

[153] R. K. Mohammed-Amin, S. von Mammen, and J. E. Boyd, "Arcs architectural chameleon skin," in *Proceedings of the 31st eCAADe Conference* (R. Stouffs and S. Sariyildiz, eds.), vol. 1 of *Computation and Performance*, pp. 467–475, Delft University of Technology, eCAADe and vdf Hochschulverlag AG, 2013.

[154] S. Penny, "Art and artificial life–a primer," *Systems Research*, vol. 10, no. 3, pp. 19–33, 1993.

[155] D. Jones, "Atomswarm: A framework for swarm improvisation," *Applications of Evolutionary Computing*, pp. 423–432, 2008.

[156] D. Bisig and T. Unemi, "Swarm simulations for dance performance," in *Proceedings of the 12th Generative Art International Conference*, (Politecnico di Milano University), pp. 105–114, 2009.

[157] J. Eisenmann, B. Schroeder, M. Lewis, and R. Parent, "Creating choreography with interactive evolutionary algorithms," in *Applications of Evolutionary Computation*, pp. 293–302, Springer, 2011.

[158] S. Dor, "A history of real-time strategy gameplay from decryption to prediction: Introducing the actional statement," in *History of Games International Conference Proceedings* (C. Therrien, H. Lowood, and M. Picard, eds.), (Montreal, Canada), Kinephanos, 2014.

[159] J. Linderoth, "Why gamers dont learn more: An ecological approach to games as learning environments," in *Proceedings of the 2010 International DiGRA Nordic Conference: Experiencing Games: Games, Play, and Players*, (Stockholm), Digra, 2010.

[160] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *Human-Machine Systems, IEEE Transactions on*, vol. 46, pp. 9–26, Feb 2016.

[161] J. P. Gee and M. H. Levine, "Welcome to our virtual worlds," *Literacy*, vol. 2, pp. 48–52, 2009.

[162] W. Ketter, J. Collins, and P. Reddy, "Power tac: A competitive economic simulation of the smart grid," *Energy Economics*, vol. 39, pp. 262–270, 2013.

[163] M. Kuit, I. S. Mayer, and M. De Jong, "The infrastratego game: An evaluation of strategic behavior and regulatory regimes in a liberalizing electricity market," *Simulation & Gaming*, vol. 36, no. 1, pp. 58–74, 2005.

[164] C. Vasile, A. Pavel, and C. Buiu, "Integrating human swarm interaction in a distributed robotic control system," in *Automation Science and Engineering (CASE), 2011 IEEE Conference on*, pp. 743–748, IEEE, 2011.

[165] J. Nagi, H. Ngo, A. Giusti, L. M. Gambardella, J. Schmidhuber, and G. A. Di Caro, "Incremental learning using partial feedback for gesture-based human-swarm interaction," in *RO-MAN, 2012 IEEE*, pp. 898–905, IEEE, 2012.

[166] S. Bashyal and G. K. Venayagamoorthy, "Human swarm interaction for radiation source search and localization," in *Swarm Intelligence Symposium, 2008. SIS 2008. IEEE*, pp. 1–8, IEEE, 2008.

[167] S. Stepney and P. S. Andrews, "Cosmos special issue editorial," *Natural Computing*, vol. 14, no. 1, pp. 1–6, 2015.

[168] J. Pitt and A. Artikis, "Engineering organised adaptation: A tutorial," in *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*, pp. 239–248, IEEE, 2012.

[169] C. Haubeck, A. Vilenica, and W. Lamersdorf, "Using building blocks for pattern-based simulation of self-organising systems," in *Intelligent Distributed Computing VI*, pp. 9–15, Springer, 2013.

[170] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221, 2012.

[171] J. Gregory, *Game engine architecture*. CRC Press, 2009.

[172] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, pp. 153:1–153:12, 2014.

[173] A. Elliott, B. Peiris, and C. Parnin, "Virtual reality in software engineering: Affordances, applications, and challenges," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2, pp. 547–550, IEEE, 2015.

[174] W. Kurth, G. Buck-Sorlin, and O. Kniemeyer, "Relationale wachstumsgrammatiken: Ein formalismus zur spezifikation multiskalierter struktur-funktions-modelle von pflanzen," in *Modellierung pflanzlicher Systeme aus historischer und aktueller Sicht.*, vol. 7 of *Landwirtschaft*, pp. 36–45, Landesamtes für Verbraucherschutz, Landwirtschaft und Flurneuordnung Brandenburg, 2006.

[175] O. Kniemeyer, G. Buck-Sorlin, and W. Kurth, "Groimp as a platform for functional-structural modelling of plants," in *Functional-Structural Plant Modelling in Crop Production* (J. Vos, L. F. M. Marcelis, P. H. B. deVisser, P. C. Struik, and J. B. Evers, eds.), pp. 43–52, Springer, March 2006.

[176] O. Kniemeyer, G. Barczik, R. Hemmerling, and W. Kurth, "Relational Growth Grammars—A Parallel Graph Transformation Approach with Applications in Biology and Architecture," in *Applications of Graph Transformations with Industrial Relevance: Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers*, (Berlin, Heidelberg), pp. 152–167, Springer-Verlag, 2008.

[177] H. Ehrig, C. Ermel, U. Golas, and F. Hermann, "Model transformation," in *Graph and Model Transformation*, pp. 43–62, Springer, 2015.

[178] O. Kniemeyer and W. Kurth, "Xl4c4d-adding the graph transformation language xl to cinema 4d," *Electronic Communications of the EASST*, vol. 61, 2013.

[179] H. Belhaouari, A. Arnould, P. Le Gall, and T. Bellet, "Jerboa: A graph transformation library for topology-based geometric modeling," in *Graph Transformation*, pp. 269–284, Springer, 2014.

[180] M. Pedersen, A. Phillips, and G. D. Plotkin, "A high-level language for rule-based modelling," *PloS one*, vol. 10, no. 6, p. e0114296, 2015.

[181] G. Misirli, M. Cavaliere, W. Waites, M. Pocock, C. Madsen, O. Gilfellon, R. Honorato-Zimmer, P. Zuliani, V. Danos, and A. Wipat, "Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualization," *Bioinformatics*, 2015.

[182] A. Namatame and S.-H. Chen, *Agent Based Modelling and Network Dynamics*. Oxford University Press, 2016.

[183] J. Chhatwal and T. He, "Economic evaluations with agent-based modelling: an introduction," *Pharmacoeconomics*, vol. 33, no. 5, pp. 423–433, 2015.

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

[185] T. Mullen, *Mastering blender*. Sybex, 2009.

[186] 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, (Perth, Australia), pp. 273–280, ACM Press, December 2007.

[187] R. Shah, *Master the Art of Unreal Engine 4 Blueprints*. Kitatus Studios, 2014.

[188] Antares Universe, "Antares universe vizio." Published online `http://www.antares-universe.com/`, March 2013.

[189] S. Machkovech, "Epic announces vr updates for unreal editor, predicts vr editing future," *ars technica*, p. published online, February 2016.

[190] J. Whyte, N. Bouchlaghem, A. Thorpe, and R. McCaffer, "From cad to virtual reality: modelling approaches, data exchange and interactive 3d building design tools," *Automation in construction*, vol. 10, no. 1, pp. 43–55, 2000.

[191] T. M. Takala, M. Makarainen, and P. Hamalainen, "Immersive 3d modeling with blender and off-the-shelf hardware," in *3D User Interfaces (3DUI), 2013 IEEE Symposium on*, pp. 191–192, IEEE, 2013.

[192] M. Mine, A. Yoganandan, and D. Coffey, "Making vr work: building a real-world immersive modeling application in the virtual world," in *Proceedings of the 2nd ACM symposium on Spatial user interaction*, pp. 80–89, ACM, 2014.

[193] D. Kornhauser, U. Wilensky, and W. Rand, "Design guidelines for agent based model visualization," *Journal of Artificial Societies and Social Simulation*, vol. 12, no. 2, pp. pp. 1–27, 2009.

[194] U. Wilensky and W. Rand, *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*. MIT Press, 2015.

[195] K. Wang, C. McCaffrey, D. Wendel, and E. Klopfer, "3d game design with programming blocks in starlogo tng," in *Proceedings of the 7th international conference on Learning sciences*, pp. 1008–1009, International Society of the Learning Sciences, 2006.

[196] 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*, (New York, NY, USA), pp. 1439–1440, ACM, 2006.

[197] R. Junges and F. Klügl, "Learning tools for agent-based modeling and simulation," *KI-Künstliche Intelligenz*, vol. 27, no. 3, pp. 273–280, 2013.

[198] M. Resnick, F. Martin, R. Sargent, and B. Silverman, "Programmable bricks: Toys to think with," *IBM Systems Journal*, vol. 35, no. 3.4, pp. 443 –452, 1996.

[199] 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 *Proceedings of the IEEE Symposium on Visual Languages*, (Darmstadt, Germany), pp. 172 –179, IEEE, September 1995.

[200] A. Repenning, C. Smith, R. Owen, and N. Repenning, "Agentcubes: Enabling 3d creativity by addressing cognitive and affective programming challenges," in *Proceedings of*

the World Conference on Educational Multimedia, Hypermedia and Telecommunications, vol. 1, pp. 2762–2771, 2012.

[201] A. Repenning, D. C. Webb, C. Brand, F. Gluck, R. Grover, S. Miller, H. Nickerson, and M. Song, "Beyond minecraft: Facilitating computational thinking through modeling and programming in 3d," *IEEE Computer Graphics and Applications*, no. 3, pp. 68–71, 2014.

[202] D. A. Tietze, *A Framework for Developing Component-based Co-operative Applications*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2000.

[203] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riß, C. Sandor, and M. Wagner, "Design of a component-based augmented reality framework," in *International Symposium on Augmented Reality*, (Los Alamitos, CA, USA), p. 45, 2001.

[204] C. Stoy, *Game Programming Gems 6*, ch. Game Object Component System, pp. 393–403. Charles River Media, 2006.

[205] J. Martin, "Entity systems are the future of mmog development." Published online `http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/`, September 2007.

[206] S. D. Peckham, E. W. Hutton, and B. Norris, "A component-based approach to integrated modeling in the geosciences: The design of csdms," *Computers & Geosciences*, vol. 53, pp. 3–12, 2013.

[207] K. Wilson, "Game object structure: Inheritance vs. aggregation." Published online `http://gamearchitect.net/Articles/GameObjects1.html`, July 2002.

[208] S. M. Lewandowski, "Frameworks for component-based client/server computing," *ACM Comput. Surv.*, vol. 30, no. 1, pp. 3–27, 1998.

[209] M. Mikic-Rakic, S. Malek, and N. Medvidovic, "Improving availability in large, distributed component-based systems via redeployment," *Component Deployment*, pp. 83–98, 2005.

[210] H. Roussain and F. Guidec, "Cooperative component-based software deployment in wireless ad hoc networks," *Component Deployment*, pp. 1–16, 2005.

[211] D. Burkhardt, T. Ruppert, and K. Nazemi, "Towards process-oriented information visualization for supporting users," in *Interactive Collaborative Learning (ICL), 2012 15th International Conference on*, pp. 1–8, 2012.

[212] P. Hitzler and K. Janowicz, "Linked data, big data, and the 4th paradigm.," *Semantic Web*, vol. 4, no. 3, pp. 233–235, 2013.

[213] S. von Mammen, C. Grenz, J. Hähner, S. Timpf, D. Loebenberger, S. Mandl, and O. Kozachuk, "Organic data: Ein sicheres, dezentralisiertes big data konzept," in *Jahrestagung der Gesellschaft für Informatik (Workshop CPSData). GI-Workshop "Technologien zur Analyse und Steuerung komplexer cyber-physischer Systeme" (CPSData-2014), located at INFORMATIK 2014: Big Data - Komplexität meistern - 44. Jahrestagung der Gesellschaft für Informatik, September 23, Stuttgart, Germany*, Springer, 2014.

[214] A. Kemper, T. Mühlbauer, T. Neumann, A. Reiser, and W. Rödiger, "Bericht vom herbsttreffen der gi-fachgruppe datenbanksysteme.," *Datenbank-Spektrum*, vol. 13, no. 1, pp. 65–66, 2013.

[215] J.-D. Cryans, A. April, and A. Abran, *Criteria to Compare Cloud Computing with Current Database Technology*, vol. 5338, pp. 114–126. Springer Berlin Heidelberg, 2008.

[216] K. Rohloff and R. E. Schantz, "High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triple-store," in *Programming Support Innovations for Emerging Distributed Applications*, PSI EtA '10, (New York, NY, USA), pp. 4:1–4:5, ACM, 2010.

[217] H. Mühleisen, T. Walther, and R. Tolksdorf, "Data Location Optimization for a Self-Organized Distributed Storage System," in *Proeedings of the Third World Congress on Nature and Biologically Inspired Computing*, pp. 176–182, IEEE Press, 2011.

[218] A. Forestiero, E. Leonardi, C. Mastroianni, and M. Meo, "Self-Chord: A Bio-Inspired P2P Framework for Self-Organizing Distributed Systems," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 1651–1664, Oct. 2010.

[219] J. Dykes, A. MacEachren, and M.-J. Kraak, *Exploring Geovisualization*. Elsevier Pergamon Press, Amsterdam, 2005.

[220] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual analytics: Definition, process, and challenges," in *Information Visualization* (A. Kerren, J. Stasko, J.-D. Fekete, and C. North, eds.), vol. 4950 of *Lecture Notes in Computer Science*, pp. 154–175, Springer Berlin Heidelberg, 2008.

[221] G. Andrienko and N. Andrienko, "Interactive maps for visual data exploration," *International Journal for Geographical Information Science*, vol. 13, no. 4, pp. 355–374, 1999.

[222] M. Ward, G. G. Grinstein, and D. Keim, *Interactive data visualization: Foundations, techniques, and applications*. AK Peters, 2010.

[223] S. Card, J. Mackinlay, and B. Shneiderman, *Readings in Information Visualization. Using Vision to think*. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[224] A. J. Lotka, "Contribution to the theory of periodic reaction," *J. Phys. Chem.*, vol. 14, no. 3, pp. 271–274, 1910.

[225] V. Volterra, "Variazioni e fluttuazioni del numero d'individui in specie animali conviventi," *Mem. Acad. Lincei. Roma*, vol. 2, pp. 31–113, 1926.

[226] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings., IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[227] A. Schiendorfer, G. Anders, J.-P. Steghöfer, and W. Reif, "Abstraction of heterogeneous supplier models in hierarchical resource allocation," in *Transactions on Computational Collective Intelligence XX*, pp. 23–53, Springer, 2015.

[228] H. V. D. Parunak and S. A. Brueckner, "Software engineering for self-organizing systems," *The Knowledge Engineering Review*, vol. 30, no. 04, pp. 419–434, 2015.

[229] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity, New York: Oxford University Press, 1999.

[230] 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: Princeton University Press, 2003.

[231] N. Napp and R. Nagpal, "Distributed amorphous ramp construction in unstructured environments," *Robotica*, vol. 32, no. 02, pp. 279–290, 2014.

[232] T. Soleymani, V. Trianni, M. Bonani, F. Mondada, and M. Dorigo, "Bio-inspired construction with mobile robots and compliant pockets," *Robotics and Autonomous Systems*, vol. 74, pp. 340–350, 2015.

[233] M. Prokopenko, *Guided self-organization: Inception*, vol. eBook. Berlin / Heidelberg: Springer, 2014.

[234] M. Prokopenko, D. Polani, and N. Ay, "On the cross-disciplinary nature of guided self-organisation," in *Guided Self-Organization: Inception*, pp. 3–15, Springer, 2014.

[235] M. Prokopenko, L. Barnett, M. Harré, J. T. Lizier, O. Obst, and X. R. Wang, "Fisher transfer entropy: quantifying the gain in transient sensitivity," in *Proc. R. Soc. A*, vol. 471, p. 20150610, The Royal Society, 2015.

[236] C. Salge, C. Glackin, and D. Polani, "Empowerment–an introduction," in *Guided Self-Organization: Inception*, pp. 67–114, Springer, 2014.

[237] N. Ay and K. Zahedi, "On the causal structure of the sensorimotor loop," in *Guided Self-Organization: Inception*, pp. 261–294, Springer, 2014.

[238] C. Gros, "Generating functionals for guided self-organization," in *Guided Self-Organization: Inception*, pp. 53–66, Springer, 2014.

[239] H. Sayama, "Guiding designs of self-organizing swarms: Interactive and automated approaches," in *Guided Self-Organization: Inception*, pp. 365–387, Springer, 2014.

[240] B. G. Bezirtzis, M. Lewis, and C. Christeson, "Interactive evolution for industrial design," in *Conference on creativity & cognition*, (New York, NY, USA), pp. 183–192, ACM, 2007.

[241] E. Bonabeau, "Agent-based modeling: methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99 Suppl 3, pp. 7280–7287, May 2002.

[242] D. Helbing and S. Balietti, *Social Self-Organization*, ch. Agent-Based Modeling, pp. 25–70. Springer Berlin Heidelberg, 2012.

[243] A. Crooks, C. Castle, and M. Batty, "Key challenges in agent-based modelling for geo-spatial simulation," *Computers, Environment and Urban Systems*, vol. 32, pp. 417–430, Nov. 2008.

[244] D. Pawlaszczyk and S. Strassburger, "Scalability in distributed simulations of agent-based models," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pp. 1189–1200, IEEE, 2009.

[245] F. Amigoni and V. Schiaffonati, "Multiagent-Based Simulation in Biology A Critical Analysis," *Model-Based Reasoning in Science, Technology, and Medicine*, vol. 64, pp. 179–191, June 2007.

[246] G. Desmeulles, G. Querrec, P. Redou, S. Kerdelo, L. Misery, V. Rodin, and J. Tisseau, "The virtual reality applied to biology understanding: The in virtuo experimentation," *Expert Systems With Applications*, vol. 30, no. 1, pp. 82–92, 2006.

[247] A. Sarraf Shirazi, S. von Mammen, and C. Jacob, "Hierarchical self-organized learning in agent-based modeling of the mapk signaling pathway," in *CEC 2011, IEEE Congress on Evolutionary Computation*, (New Orleans, Louisiana), pp. 2245–2251, IEEE Press, 2011.

[248] H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *Visualization'98. Proceedings*, pp. 35–42, IEEE, 1998.

[249] P. P. Rodrigues, J. Gama, and J. P. Pedroso, "Hierarchical clustering of time-series data streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 20, no. 5, pp. 615–627, 2008.

[250] S. Rudolph, S. Tomforde, B. Sick, H. Heck, A. Wacker, and J. Haehner, "An online influence detection algorithm for organic computing systems," in *Architecture of Computing Systems. Proceedings, ARCS 2015-The 28th International Conference on*, pp. 1–8, VDE, 2015.

[251] S. Rudolph, S. Tomforde, B. Sick, and J. Hähner, "A mutual influence detection algorithm for systems with local performance measurement," in *Proceedings of IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, (Cambridge, MA, USA), to appear.

[252] B. Labbe, R. Hérault, and C. Chatelain, "Learning deep neural networks for high dimensional output problems," in *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*, pp. 63–68, IEEE, 2009.

[253] L. Arnold, H. Paugam-Moisy, and M. Sebag, "Unsupervised layer-wise model selection in deep neural networks," in *19th European Conference on Artificial Intelligence (ECAI'10)*, 2010.

[254] J. Gehring, Y. Miao, F. Metze, and A. Waibel, "Extracting deep bottleneck features using stacked auto-encoders," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 3377–3381, IEEE, 2013.

[255] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 59, no. 4-5, pp. 291–294, 1988.

[256] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[257] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[258] K. Chen, "Deep and modular neural networks," in *Springer Handbook of Computational Intelligence*, pp. 473–494, Springer, 2015.

[259] M. Scheffer, J. M. Baveco, D. L. DeAngelis, K. A. Rose, and E. H. van Nes, "Super-individuals a simple solution for modelling large populations on an individual basis," *Ecological modelling*, vol. 80, no. 1995, pp. 161–170, 1995.

[260] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[261] P. Schuster, "How does complexity arise in evolution," *Complex.*, vol. 2, pp. 22–30, Sept. 1996.

[262] A. Dyonizy, V. Kaminker, J. Wieckowska, T. Krzywicki, J. Pantaleone, P. Nowak, and J. Maselko, "Cyclic growth of hierarchical structures in the aluminum-silicate system," *Journal of systems chemistry*, vol. 6, no. 1, p. 3, 2015.

[263] M. Bosse, A. Heuwieser, A. Heinzel, I. Nancucheo, H. M. B. Dall'Agnol, A. Lukas, G. Tzotzos, and B. Mayer, "Interaction networks for identifying coupled molecular processes in microbial communities," *BioData mining*, vol. 8, no. 1, pp. 1–17, 2015.

[264] A. Dorin and J. McCormack, "Self-assembling dynamical hierarchies," *Artificial life eight*, p. 423, 2003.

[265] M. Mnif and C. Müller-Schloer, "Quantitative emergence," in *Proceedings of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (SMCals 2006)*, (Piscataway, NJ, USA), pp. 78–84, IEEE, July 2006.

[266] D. Fisch, M. Jänicke, B. Sick, and C. Müller-Schloer, "Quantitative emergence–a refined approach based on divergence measures," in *Self-Adaptive and Self-Organizing Systems (SASO), 2010 4th IEEE International Conference on*, pp. 94–103, IEEE, 2010.

[267] D. Fisch, M. Jänicke, E. Kalkowski, and B. Sick, "Techniques for knowledge acquisition in dynamically changing environments," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 7, no. 1, p. 16, 2012.

[268] S. von Mammen, J.-P. Steghöfer, J. Denzinger, and C. Jacob, "Self-organized middle-out abstraction," in *Self-Organizing Systems* (C. Bettstetter and C. Gershenson, eds.), vol. 6557 of *Lecture Notes in Computer Science*, (Karslruhe, Germany), pp. 26–31, Springer Verlag, 2011.

[269] M. Horstemeyer, "Multiscale modeling: A review," *Practical Aspects of Computational Chemistry*, pp. 87–135, 2010.

[270] N. Baas and C. Emmeche, "On emergence and explanation," *Intellectica*, no. 2, no. 25, pp. 67–83, 1997.

[271] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 9, no. 4, pp. 586–600, 2015.

[272] S. Gite and H. Agrawal, "On context awareness for multisensor data fusion in iot," in *Proceedings of the Second International Conference on Computer and Communication Technologies*, pp. 85–93, Springer, 2016.

[273] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *The Knowledge Engineering Review*, vol. 19, no. 04, pp. 281–316, 2004.

[274] H. A. Abbas, S. I. Shaheen, and M. H. Amin, "Organization of multi-agent systems: An overview," *Journal of Intelligent Information Systems*, vol. 4, no. 3, pp. 46–57, 2015.

[275] A. Koestler, *The Ghost in the Machine*. Macmillan, 1968.

[276] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux, "Dynamic self-organization in holonic multi-agent manufacturing systems: The adacor evolution," *Computers in Industry*, vol. 66, pp. 99–111, 2015.

[277] M. Eigen and P. Schuster, "The hypercycle," *Naturwissenschaften*, vol. 65, no. 1, pp. 7–41, 1978.

[278] J.-L. Dessalles and D. Phan, "Emergence in multi-agent systems: cognitive hierarchy, detection, and complexity reduction part i: methodological issues," in *Artificial Economics*, pp. 147–159, Springer, 2006.

[279] J. Denzinger and J. Hamdan, "Improving observation-based modeling of other agents using tentative stereotyping and compactification through kd-tree structuring," *Web Intelligence and Agent Systems*, vol. 4, pp. 255–270, 2006.

[280] Y. Bachrach, A. Parnes, A. D. Procaccia, and J. S. Rosenschein, "Gossip-based aggregation of trust in decentralized reputation systems," *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 2, pp. 153–172, 2009.

[281] A. Mosebach and J. Lunze, "A deterministic gossiping algorithm for the synchronization of multi-agent systems," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 1–7, 2015.

[282] The Presidents and Fellows of Harvard College, "Bio{V}isions at {H}arvard {U}niversity." Published online `http://multimedia.mcb.harvard.edu/`, February 2016.

[283] M. J. Ackerman, T. Yoo, and D. Jenkins, "From data to knowledge - the visible human project®continues," *MEDINFO 2001*, pp. 887–890, 2001.

[284] M. F. Seifert, "Visible human projects special issue," *Clinical Anatomy*, vol. 19, no. 3, pp. 191–289, 2010.

[285] N. Mitsuhashi, K. Fujieda, T. Tamura, S. Kawamoto, T. Takagi, and K. Okubo, "BodyParts3D: 3D structure database for anatomical concepts," *Nucl. Acids Res.*, p. gkn613, 2008.

[286] C. W. Sensen and J. Soh, *Advanced Imaging in Biology and Medicine*, vol. II, ch. CAVEman, An Object-Oriented Model of the Human Body, pp. 289–300. Berlin, Heidelberg: Springer, 2009.

[287] P. J. Hunter and T. K. Borg, "Integration from proteins to organs: the physiome project," *Nature Reviews Molecular Cell Biology*, vol. 4, no. 3, pp. 237–243, 2003.

[288] I. Burleigh, G. Suen, and C. Jacob, "Dna in action! a 3d swarm-based model of a gene regulatory system," in *ACAL 2003, First Australian Conference on Artificial Life*, (Canberra, Australia), 2003.

[289] C. Jacob, A. Barbasiewicz, and G. Tsui, "Swarms and Genes: Exploring λ-Switch Gene Regulation through Swarm Intelligence," in *CEC 2006, IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada*, 2006.

[290] I. Yazdanbod and S. Marcus, "An agent-based simulation of blood coagulation processes," *The Journal of Undergraduate Research in Alberta*, vol. 1, no. 1, pp. 13–18, 2010.

[291] G. T. Heineman and W. T. Councill, *Component-Based Software Engineering*. ACM Press, 2001.

[292] G. T. Leavens and M. Sitaraman, eds., *Foundations of component-based systems*. New York, NY, USA: Cambridge University Press, 2000.

[293] K. Årzén, A. Bicchi, G. Dini, S. Hailes, K. Johanesson, J. Lygeros, and A. Tzes, "A component-based approach to the design of networked control systems," *European Journal of Control*, vol. 13, no. 2, pp. 261–279, 2007.

[294] E. de Lara, D. S. Wallach, and W. Zwaenepoel, "Puppeteer: Component-based adaptation for mobile computing," in *USENIX Symposium on Internet Technologies and Systems*, 2000.

[295] Zygote Inc., "Zygote human anatomy 3d model." Published online `http://www.zygote.com/`, April 2010.

[296] COLLADA.org, "Collada - digital asset and fx exchange schema." http://www.collada.org, 04 2010.

[297] N. L. of Medicine, "National library of medicine: The visible human project." Published online `http://www.nlm.nih.gov/research/visible/visible_human.html`, 04 2010.

[298] S. Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. New York: Scribner, 2001.

[299] S. Wolfram, *A new kind of science*. Champaign, Ilinois, US, United States: Wolfram Media Inc., 2002.

[300] M. Pogson, R. Smallwood, E. Qwarnstrom, and M. Holcombe, "Formal agent-based modelling of intracellular chemical interactions," *BioSystems*, 2006.

[301] D. C. Walker and J. Southgate, "The virtual cell–a candidate co-ordinator for 'middle-out' modelling of biological systems," *Briefings in Bioinformatics*, vol. 10, pp. 450–461, 7 2009.

[302] R. Hoar, J. Penner, and C. Jacob, "Transcription and evolution of a virtual bacteria culture," in *IEEE Congress on Evolutionary Computation*, (Canberra, Australia), IEEE Press, 2003.

[303] J. Penner, R. Hoar, and C. Jacob, "Bacterial chemotaxis in silico," in *ACAL 2003, First Australian Conference on Artificial Life*, (Canberra, Australia), 2003.

[304] C. Jacob and I. Burleigh, "Biomolecular Swarms: An Agent-based Model of the Lactose Operon," *Natural Computing*, vol. 3, pp. 361–376, Dec. 2004.

[305] S. von Mammen, T. Davison, H. Baghi, and C. Jacob, "Component-based networking for simulations in medical education," in *IEEE Symposium on Computers and Communications*, (Riccione, Italy), pp. 975–979, IEEE Computer Society Washington, IEEE Press, 2010.

[306] G. E. Buchholz, W. Eßer, and J. Fedderwitz, "Geschäftsbericht 2013/2014." Annual Report of Kassenzahnärztliche Bundesvereinigung (KZBV), 2014.

[307] S. Friedman, "Prognosis of initial endodontic therapy," *Endodontic Topics*, vol. 2, no. 1, pp. 59–88, 2002.

[308] J. Siqueira, "Aetiology of root canal treatment failure: why well-treated teeth can fail," *International Endodontic Journal*, vol. 34, no. 1, pp. 1–10, 2001.

[309] G. B. Leoni, M. A. Versiani, J. D. Pécora, and M. Damião de Sousa-Neto, "Micro–computed tomographic analysis of the root canal morphology of mandibular incisors," *Journal of endodontics*, vol. 40, no. 5, pp. 710–716, 2014.

[310] C. C. d. S. A. Lins, E. M. V. de Melo Silva, G. A. de Lima, S. E. A. C. de Menezes, and R. M. C. Travassos, "Operating microscope in endodontics: A systematic review," *Open Journal of Stomatology*, vol. 3, p. 1, 2013.

[311] F. J. Vertucci, "Root canal morphology and its relationship to endodontic procedures," *Endodontic topics*, vol. 10, no. 1, pp. 3–29, 2005.

[312] J. I. Ingle and H. C. Slavkin, *Ingle's Endodontics 6*, ch. Modern endodontic therapy: Past, present and future, pp. 1–35. BC Decker Inc, 6 ed., 2008.

[313] I. Marras, N. Nikolaidis, G. Mikrogeorgis, K. Lyroudia, and I. Pitas, "A virtual system for cavity preparation in endodontics," *Journal of dental education*, vol. 72, no. 4, pp. 494–502, 2008.

[314] S. Suebnukarn, R. Hataidechadusadee, N. Suwannasri, N. Suprasert, P. Rhienmora, and P. Haddawy, "Access cavity preparation training using haptic virtual reality and micro-computed tomography tooth models," *International endodontic journal*, vol. 44, no. 11, pp. 983–989, 2011.

[315] A. Urbankova and R. Lichtenthal, "Dentsim virtual reality in preclinical operative dentistry to improve psychomotor skills–a pilot study," *Journal of Dental Education*, vol. 66, no. 2, pp. 284–292, 2002.

[316] J. Forsslund, "Simulator for operative extraction of wisdom teeth," *SIGRAD 2008*, p. 23, 2008.

[317] A. D. Steinberg, P. G. Bashook, J. Drummond, S. Ashrafi, and M. Zefran, "Assessment of faculty perception of content validity of periosim©, a haptic-3d virtual reality dental training simulator," *Journal of Dental Education*, vol. 71, no. 12, pp. 1574–1582, 2007.

[318] M. M. Bakr, W. Massey, and H. Alexander, "Students' evaluation of a 3dvr haptic device (simodont®). does early exposure to haptic feedback during preclinical dental education enhance the development of psychomotor skills?," *International Journal of Dental Clinics*, vol. 6, no. 2, 2014.

[319] K. Erleben, J. Sporring, K. Henriksen, and H. Dohlmann, *Physics-based animation*. Charles River Media Hingham, 2005.

[320] J. Bender, K. Erleben, J. Trinkle, and E. Coumans, "Interactive simulation of rigid body dynamics in computer graphics," *STAR Proceedings of Eurographics*, 2012.

[321] R. Featherstone and D. Orin, "Robot dynamics: Equations and algorithms," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1, pp. 826–834, IEEE, 2000.

[322] R. Diziol, J. Bender, and D. Bayer, "Robust real-time deformation of incompressible surface meshes," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 237–246, ACM, 2011.

[323] M. Liu and G. Liu, "Smoothed particle hydrodynamics (sph): an overview and recent developments," *Archives of computational methods in engineering*, vol. 17, no. 1, pp. 25–76, 2010.

[324] R. W. Batterman, "Falling cats, parallel parking, and polarized light," *Studies In History and Philosophy of Modern Physics*, vol. 34, no. 4, pp. 527–557, 2003.

[325] S. F. Gibson and B. Mirtich, "A survey of deformable modeling in computer graphics," tech. rep., Mitsubishi Electric Research Laboratories, 1997.

[326] D. Wang, Y. Zhang, Y. Wang, P. Lü, R. Zhou, and W. Zhou, "Haptic rendering for dental training system," *Science in China Series F: Information Sciences*, vol. 52, no. 3, pp. 529–546, 2009.

[327] A. Fuhrmann, G. Sobotka, and C. Groß, "Distance fields for rapid collision detection in physically based modeling," in *Proceedings of GraphiCon 2003*, pp. 58–65, 2003.

[328] K. Lundin, A. Ynnerman, and B. Gudmundsson, "Proxy-based haptic feedback from volumetric density data," in *Proceedings of the Eurohaptic Conference*, pp. 104–109, 2002.

[329] A. Lécuyer, "Simulating haptic feedback using vision: A survey of research and applications of pseudo-haptic feedback," *Presence: Teleoperators and Virtual Environments*, vol. 18, no. 1, pp. 39–53, 2009.

[330] R. Malladi and J. A. Sethian, "A real-time algorithm for medical shape recovery," in *Computer Vision, 1998. Sixth International Conference on*, pp. 304–310, IEEE, 1998.

[331] K. Krissian and C.-F. Westin, "Fast sub-voxel re-initialization of the distance map for level set methods," *Pattern Recognition Letters*, vol. 26, no. 10, pp. 1532–1542, 2005.

[332] O. Sharma, Q. Zhang, F. Anton, and C. Bajaj, "Fast streaming 3d level set segmentation on the gpu for smooth multi-phase segmentation," in *Transactions on computational science XIII*, pp. 72–91, Springer, 2011.

[333] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of SIGGRAPH '87*, pp. 163–169, ACM, 1987.

[334] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," in *Proceedings of SIGGRAPH '02*, pp. 339–346, ACM, 2002.

[335] F. A. C. Polack, P. S. Andrews, T. Ghetiu, M. Read, S. Stepney, J. Timmis, and A. T. Sampson, "Reflections on the simulation of complex systems for science," in *Fifteenth IEEE International Conference on Engineering of Complex Computer Systems, ICECCS'10* (R. Calinescu, R. Paige, and M. Kwiatkowska, eds.), (University of Oxford, UK), pp. 276–285, IEEE Press, Mar. 2010.

[336] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2010.

[337] W. Banzhaf, "Artificial chemistries - Towards Constructive Dynamical Systems," *Solid State Phenomena*, pp. 43–50, 2004.

[338] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer-Verlag, 1996.

[339] N. Chomsky, "Three models for the description of language," *Information Theory, IRE Transactions on*, vol. 2, no. 3, pp. 113–124, 1956.

[340] J. von Neumann and A. W. Burks, *Theory of self-reproducing automata*. Urbana and London: University of Illinois Press, 1966.

[341] S. Kauffman, *The origins of order: Self-Organisation and Selection in Evolution*. Oxford Univ. Press New York, 1993.

[342] I. Salazar-Ciudad, "Tooth Morphogenesis in vivo, in vitro, and in silico," *Current Topics in Developmental Biology*, vol. 81, p. 342, 2008.

[343] C. Jacob and S. von Mammen, "Swarm grammars: growing dynamic structures in 3d agent spaces," *Digital Creativity: Special issue on Computational Models of Creativity in the Arts*, vol. 18, pp. 54–64, March 2007.

[344] S. von Mammen and C. Jacob, "Genetic swarm grammar programming: Ecological breeding like a gardener," in *CEC 2007, IEEE Congress on Evolutionary Computation* (D. Srinivasan and L. Wang, eds.), IEEE Press, (Singapore), pp. 851–858, 2007.

[345] S. von Mammen and C. Jacob, "Evolutionary swarm design of architectural idea models," in *Genetic and Evolutionary Computation Conference (GECCO) 2008*, (Atlanta, USA), pp. 143–150, ACM Press, 2008.

[346] S. von Mammen, D. Phillips, T. Davison, and C. Jacob, "Swarm graph grammars: A scale-independent, graph-based multi-agent representation," in *ANTS 2010: Seventh International Conference on Swarm Intelligence*, Springer, 2010.

[347] S. von Mammen, J. Wong, and C. Jacob, "Virtual constructive swarms: Compositions and inspirations," in *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2008*, vol. 4974 of *Lecture Notes in Computer Science*, (Berlin-Heidelberg), pp. 491–496, Springer-Verlag, 2008.

[348] S. von Mammen and C. Jacob, "Swarm-driven idea models - from insect nests to modern architecture," in *Eco-Architecture 2008, Second International Conference on Harmonisation Between Architecture and Nature* (C. Brebbia, ed.), (Winchester, UK), pp. 117–126, WIT Press, 2008.

[349] S. von Mammen, T. Wissmeier, J. Wong, and C. Jacob, "Artistic exploration of the worlds of digital developmental swarms," *LEONARDO*, vol. 44, pp. 5–13, January 2011.

[350] E. McKean, ed., *The New Oxford American Dictionary*. Oxford University Press, 2005.

[351] D. Hu and R. Marcucio, "A SHH-responsive signaling center in the forebrain regulates craniofacial morphogenesis via the facial ectoderm," *Development*, vol. 136, no. 1, p. 107, 2009.

[352] B. Hallgrímsson, J. C. Boughner, A. Turinsky, T. E. Parsons, C. Logan, and C. W. Sensen, "Geometric morphometrics and the study of development," *Advanced Imaging in Biology and Medicine*, pp. 319–336, 2009.

[353] G. S. Hornby, "Measuring complexity by measuring structure and organization," in *2007 IEEE Congress on Evolutionary Computation* (D. Srinivasan and L. Wang, eds.), (Singapore), pp. 2017–2024, IEEE Press, 2007.

[354] J.-L. Giavitto and O. Michel, "Modeling the topological organization of cellular processes," *Biosystems*, vol. 70, no. 2, pp. 149–163, 2003.

[355] S. Wolfram, "Cellular automata as models of complexity," *Nature*, vol. 311, pp. 419–424, October 1984.

[356] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Mech, "Visual models of plant development," in *Handbook of Formal Languages* (G. Rozenberg and A. Salomaa, eds.), New York: Springer, 1997.

[357] J. Yu, "Evolutionary design of 2d fractals and 3d plant structures for computer graphics," master's thesis, Department of Computer Science, University of Calgary, 2004.

[358] R. Mech and P. Prusinkiewicz, "Visual models of plants interacting with their environment," in *SIGGRAPH'96*, (New Orleans, Louisiana), pp. 397–410, ACM SIGGRAPH, New York, 1996.

[359] M. T. Michalewicz, ed., *Plants to Ecosystems: Advances in Computational Life Sciences*. Collingwood, VIC, Australia: CSIRO Publishing, 1997.

[360] O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems," in *SIGGRAPH 98, Computer Graphics, Annual Conference Series*, pp. 275–286, ACM SIGGRAPH, 1998.

[361] C. Jacob, "Genetic l-system programming," in *PPSN III - Parallel Problem Solving from Nature*, vol. 866 of *Lecture Notes in Computer Science*, (Jerusalem, Israel), pp. 334–343, Springer, 1994.

[362] C. Jacob, "Evolving evolution programs: Genetic programming and l-systems," in *Genetic Programming 1996: First Annual Conference* (J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. Riolo, eds.), (Stanford University, Palo Alto, CA), pp. 107–115, MIT Press, Cambridge, MA, 1996.

[363] C. Jacob, "Evolution and co-evolution of developmental programs," *Computer Physics Communications, Special Issue, Modeling Collective Phenomena in the Sciences*, 1999.

[364] K. J. Mock, "Wildwood: The evolution of l-system plants for virtual environments," in *IEEE Conference on Evolutionary Computation*, (Anchorage, AL), pp. 476–480, IEEE Press, New York, 1998.

[365] C. Jacob, *Illustrating Evolutionary Computation with Mathematica*. San Francisco, CA: Morgan Kaufmann Publishers, 2001.

[366] F. Michel, G. Beurier, and J. Ferber, "The turtlekit simulation platform: Application to complex systems," in *Proceedings of Workshop Sessions at the 1st International Conference on Signal & Image Technology and Internet-Based Systems (IEEE SITIS05)*, pp. 122–128, IEEE Press, 2005.

[367] M. Ebner, "Coevolution and the red queen effect shape virtual plants," *Genetic Programming and Evolvable Machines*, vol. 7, no. 1, pp. 103–123, 2006.

[368] G. S. Hornby and J. B. Pollack, "Evolving l-systems to generate virtual creatures," *Computers & Graphics*, vol. 25, pp. 1041–1048, 2001.

[369] G. S. Hornby and J. B. Pollack, "Body-brain co-evolution using L-systems as a generative encoding," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, eds.), (San Francisco, California, USA), pp. 868–875, Morgan Kaufmann, 7-11 2001.

[370] G. Kókai, R. Ványi, and Z. Tóth, "Parametric l-system description of the retina with combined evolutionary operators," in *Genetic and Evolutionary Computation Conference, GECCO-99*, (Orlando, Florida, USA), 1999.

[371] G. Kókai, Z. Tóth, and R. Ványi, "Modelling blood vessel of the eye with parametric l-systems using evolutionary algorithms," in *Artificial Intelligence in Medicine, Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making, AIMDM'99* (W. Horn, Y. Shahar, G. Lindberg, S. Andreassen, and J. C. Wyatt, eds.), vol. 1620, pp. 433–443, 1999.

[372] M. Settles, P. Nathan, and T. Soule, "Breeding swarms: a new approach to recurrent neural network training," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, (New York, NY, USA), pp. 185–192, ACM Press, 2005.

[373] M. Settles and T. Soule, "Breeding swarms: a ga/pso hybrid," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, (New York, NY, USA), pp. 161–168, ACM Press, 2005.

[374] T. Blackwell, "Swarming and music," *Evolutionary Computer Music*, pp. 194–217, 2007.

[375] L. Spector, J. Klein, C. Perry, and M. Feinstein, "Emergence of collective behavior in evolving populations of flying agents," in *Genetic and Evolutionary Computation Conference (GECCO-2003)*, (Chicago, IL), pp. 61–73, Springer-Verlag, 2003.

[376] H. Kwong and C. Jacob, "Evolutionary exploration of dynamic swarm behaviour," in *Congress on Evolutionary Computation*, (Canberra, Australia), IEEE Press, 2003.

[377] B. Hölldobler and E. O. Wilson, *The Ants*. Berlin-Heidelberg: Springer-Verlag, 1990.

[378] S. von Mammen, C. Jacob, and G. Kókai, "Evolving swarms that build 3d structures," in *CEC 2005, IEEE Congress on Evolutionary Computation*, (Edinburgh, UK), pp. 1434–1441, IEEE Press, 2005.

[379] S. Van der Ryn and S. Cowan, *Ecological Design*. Island Press, 2007.

[380] G. Farmer and S. Guy, "Visions of Ventilation: Pathways to Sustainable Architecture," *Department of Architecture, University of Newcastle upon Tyne, Newcastle upon Tyne,(UK)*, 2002.

[381] K. Gowri, "Green building rating systems: An overview.," *ASHRAE Journal*, vol. 46, no. 11, pp. 56–60, 2004.

[382] T. Davison, S. von Mammen, and C. Jacob, "Evoshelf : A system for managing and exploring evolutionary data," in *Parallel Problem Solving in Nature (PPSN)*, Lecture Notes in Computer Science, pp. 310–319, Springer, 2010.

[383] C. Jacob, J. Litorco, and L. Lee, "Immunity through swarms: Agent-based simulations of the human immune system," in *Artificial Immune Systems, ICARIS 2004, Third International Conference*, (Catania, Italy), pp. 400–412, LNCS 3239, Springer, 2004.

[384] C. Jacob, S. Steil, and K. Bergmann, "The swarming body: Simulating the decentralized defenses of immunity," in *Artificial Immune Systems, ICARIS 2006, 5th International Conference*, (Oeiras, Portugal), pp. 52–65, Springer, September 2006.

[385] S. Kumar and P. Bentley, eds., *On Growth, Form and Computers*. London: Elsevier Academic Press, 2003.

[386] H. De Garis, "Artificial embryology: The genetic programming of an artificial embryo," *Dynamic, Genetic, and Chaotic Programming*, 1992.

[387] P. J. Bentley and S. Kumar, "Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, 1999.

[388] S. Kumar and P. J. Bentley, "Biologically inspired evolutionary development," *Evolvable Systems: From Biology to Hardware*, pp. 99–106, 2003.

[389] G. Beurier, F. Michel, and J. Ferber, "A morphogenesis model for multiagent embryogeny," in *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems (ALIFE X)*, 2006.

[390] R. Doursat, "Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering," *Organic Computing*, 2007.

[391] C. Smith, *On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling*. PhD thesis, University of Calgary, 2006.

[392] A. Lindenmayer, "Developmental systems without cellular interactions, their languages and grammars," *Journal of Theoretical Biology*, vol. 30, no. 3, pp. 455–484, 1971.

[393] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Connallen, and K. A. Houston, "Object-oriented analysis and design with applications, third edition," *SIGSOFT Softw. Eng. Notes*, vol. 33, pp. 11:29–11:29, Aug. 2008.

[394] P. Dittrich, J. Ziegler, and W. Banzhaf, "Artificial Chemistries - A Review," *Artificial Life*, vol. 7, pp. 225–275, 2001.

[395] I. Karsai and Z. Penzes, "Comb building in social wasps: Self-organization and stigmergic script," *Journal of Theoretical Biology*, vol. 161, no. 4, pp. 505–525, 1993.

[396] D. Erdman and D. Lee, "davidclovers." Published online `davidclovers.com`, 2010.

[397] R. Snooks and R. Stuart-Smith, "kokkugia." Published online `http://kokkugia.com/`, 2010.

[398] C. Reas, "Reas." Published online `http://reas.com`, 2010.

[399] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of SIGGRAPH '87*, (New York, NY, USA), pp. 25–34, ACM, 1987.

[400] C. J. Davidson, *Genetic Interactions in the Development of Spatial Patterns in Bacteria*. PhD thesis, University of Calgary, Calgary, Canada, 2007.

[401] H. Y. Roman Durikovic, Kazufumi Kaneda, "Animation of biological organ growth based on l-systems," *Computer Graphics Forum*, vol. 17, no. 3, pp. 1–13, 1998.

[402] M. Hemberg, U.-M. O'Reilly, A. Menges, K. Jonas, M. da Costa Gonçalves, and S. R. Fuchs, *The Art of Artificial Life: A Handbook on Evolutionary Art and Music*, ch. Genr8: Architects' Experience with an Emergent Design Tool, pp. 167–188. Natural Computing Series, Springer, 2008.

[403] P. Bentley and D. Corne, eds., *Creative Evolutionary Systems*. Artificial Intelligence, San Francisco, CA: Morgan Kaufmann, 2001.

[404] M. King, "Programmed graphics in computer art and animation," *Leonardo*, vol. 28, no. 2, pp. 113–121, 1995.

[405] J. McCormack, J. Bird, A. Dorin, and A. Jonson, *Impossible Nature: The Art of John McCormack*. Australian Centre for the Moving Image, 2004.

[406] R. Dawkins, *The Blind Watchmaker*. Harlow: Longman Scientific and Technical, 1987.

[407] K. Sims, "Artificial evolution for computer graphics," in *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, vol. 25(4), (New York), pp. 319–328, ACM Press, 1991.

[408] J. Romero and P. Machado, *The art of artificial evolution: A handbook on evolutionary art and music*. Springer-Verlag New York Inc, 2007.

[409] C. Jacob, G. Hushlak, J. Boyd, P. Nuytten, M. Sayles, and M. Pilat, "Swarmart: Interactive art from swarm intelligence," *Leonardo*, vol. 40, no. 3, 2007.

[410] G. Zugmann, P. Noever, C. Reder, C. Himmelblau, M. C. for Art, and L. A. Architecture, *Gerald Zugmann: Blue Universe: Architectural Manifestos by Coop Himmelb(l)au*. Hatje Cantz, 2002.

[411] G. Hushlak, "Website of Gerald Hushlak." Published online `http://www.ucalgary.ca/~hushlak/`, March 2009.

[412] N. Gershenfeld, *FAB: The Coming Revolution on Your Desktop–From Personal Computers to Personal Fabrication*. Basic Books, 2005.

[413] N. Bhalla and C. Jacob, "A Framework for Analyzing and Creating Self-assembling Systems," in *IEEE Swarm Intelligence Symposium, 2007. SIS 2007*, pp. 281–288, 2007.

[414] M. B. Jones, M. P. Schildhauer, O. Reichman, and S. Bowers, "The new bioinformatics: integrating ecological data from the gene to the biosphere," *Annual Review of Ecology, Evolution, and Systematics*, pp. 519–544, 2006.

[415] R. D. Stevenson, K. M. Klemow, and L. J. Gross, "Harnessing bits and bytes to transform ecology education," *Frontiers in Ecology and the Environment*, vol. 12, no. 5, pp. 306–307, 2014.

[416] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pp. 2425–2428, ACM, 2011.

[417] F. Groh, "Gamification: State of the art definition and utilization," *Institute of Media Informatics Ulm University*, pp. 39–47, 2012.

[418] AVMA, US, *Pet Ownership & Demographics Sourcebook*. Schaumberg, IL: American Veterinary Medical Association, 2012.

[419] J. Allen and M. Nelson, "Overview and design biospherics and biosphere 2, mission one (1991–1993)," *Ecological Engineering*, vol. 13, no. 1, pp. 15–29, 1999.

[420] M. Schilthuizen, "Life in little worlds," *The Loom of Life: Unravelling Ecosystems*, pp. 1–10, 2009.

[421] J. H. Bailey-Brock and R. E. Brock, "Feeding, reproduction, and sense organs of the hawaiian anchialine shrimp halocaridina rubra (atyidae)," *Pacific Science*, vol. 47, no. 4, pp. 338–355, 1993.

[422] C. S. Miller, J. F. Lehman, and K. R. Koedinger, "Goals and learning in microworlds," *Cognitive Science*, vol. 23, no. 3, pp. 305–336, 1999.

[423] A. Druin, P. Blikstein, M. Fleer, J. C. Read, B. S. Thomsen, B. D. Johnson, and M. Resnick, "How can interaction with digital creative tools support child development?:(closing panel)," in *Proceedings of the 2014 conference on Interaction design and children*, pp. 361–361, ACM, 2014.

[424] R. L. Goldstone and J. Y. Son, "The transfer of scientific principles using concrete and idealized simulations," *The Journal of the Learning Sciences*, vol. 14, no. 1, pp. 69–110, 2005.

[425] J. Tan and G. Biswas, "Simulation-based game learning environments: Building and sustaining a fish tank," in *Digital Game and Intelligent Toy Enhanced Learning, 2007. DIGITEL'07. The First IEEE International Workshop on*, pp. 73–80, IEEE, 2007.

[426] R. Vollmeyer, B. D. Burns, and K. J. Holyoak, "The impact of goal specificity on strategy use and the acquisition of problem structure," *Cognitive Science*, vol. 20, no. 1, pp. 75–100, 1996.

[427] R. Koster, *Theory of fun for game design*. O'Reilly Media, Inc., 2013.

[428] P. D. Dean L. Shumway, Charles E. Warren, "Influence of oxygen concentration and water movement on the growth of steelhead trout and coho salmon embryos," *Transactions of the American Fisheries Society, Volume 93, Issue 4, pages 342-356*, 1964.

[429] J. Brett, "The metabolic demand for oxygen in fish, particularly salmonids, and a comparison with other vertebrates," *Respiration Physiology, Volume 14, Issues 1-2, Pages 151-170*, 1972.

[430] fishbase.org, "Guppies." Published online `http://www.fishbase.org/Summary/SpeciesSummary.php?ID=3228&AT=guppy`, February 2015.

[431] fishbase.org, "Scalare." Published online `http://www.fishbase.org/Summary/speciesSummary.php?ID=4717&AT=scalare`, February 2015.

[432] planetinverts.com, "Horned nerite snail," *Published online* ***http://www.planetinverts.com/horned_nerite_snail.html***, 2015.

[433] J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, "A system of systems approach to the evolutionary transformation of power management systems.," in *GI-Jahrestagung*, pp. 1500–1515, 2013.

[434] U. Herrmann, B. Kelly, and H. Price, "Two-tank molten salt storage for parabolic trough solar power plants," *Energy*, vol. 29, no. 5, pp. 883–893, 2004.

[435] J. V. Paatero and P. D. Lund, "Effect of energy storage on variations in wind power," *Wind Energy*, vol. 8, no. 4, pp. 421–441, 2005.

[436] G. M. Masters, *Renewable and efficient electric power systems*. John Wiley & Sons, 2013.

[437] V. S. Colella, E. Klopfer, and M. Resnick, *Adventures in Modeling: Exploring Complex, Dynamic Systems with StarLogo*. New York: Teachers College Press, Columbia University, 2001.

[438] PowerWorld Corporation, "PowerWorld Simulator." Published online `http://www.powerworld.com/products/simulator/overview`, October 2014.

[439] R. C. Dugan, "Reference guide: The open distribution system simulator (opendss)," *Electric Power Research Institute, Inc*, 2012.

[440] D. Chassin, K. Schneider, and C. Gerkensmeyer, "GridLAB-D: An open-source power systems modeling and simulation environment," in *Transmission and Distribution Conference and Exposition, 2008. T&D. IEEE/PES*, pp. 1–5, IEEE, 2008. `http://www.gridlabd.org/`.

[441] J. Shearer and M. Wolfe, "Alglib, a simple symbol-manipulation package," *Communications of the ACM*, vol. 28, no. 8, pp. 820–825, 1985.

[442] ALGLIB Project, "ALGLIB: a cross-platform open source numerical analysis and data processing library." Published online `http://www.alglib.net/`, January 2014.

[443] W. Brockmann, E. Maehle, and F. Mösch, "Organic fault-tolerant control architecture for robotic applications," in *IARP/IEEE-RAS/EURON Workshop on Dependable Robots in Human Environments*, 2005.

[444] F. Mösch, M. Litza, A. E. S. Auf, E. Maehle, K.-E. Großpietsch, and W. Brockmann, "Orca - towards an organic robotic control architecture," in *Proc. of First International Workshop on Self-Organizing Systems, IWSOS/EuroNGI*, pp. 251–253, 2006.

[445] W. Brockmann, N. Rosemann, and E. Maehle, "A Framework for Controlled Self-optimisation in Modular System Architectures," in *Organic Computing - A Paradigm Shift for Complex Systems*, pp. 281 – 294, Birkhäuser Verlag, 2011.

[446] A. Jungmann, B. Kleinjohann, and W. Richert, "Increasing learning speed by imitation in multi-robot societies," in *Organic Computing - A Paradigm Shift for Complex Systems*, pp. 295–307, Birkhäuser Verlag, 2011.

[447] B. Kempkes and F. Meyer auf der Heide, "Local, self-organizing strategies for robotic formation problems," in *ALGOSENSORS*, vol. 7111 of *Lecture Notes in Computer Science*, pp. 4–12, Springer, 2011.

[448] P. Brandes, B. Degener, B. Kempkes, and F. Meyer auf der Heide, "Energy-efficient strategies for building short chains of mobile robots locally," in *SIROCCO '11: Proc. of the 18th International Colloquium on Structural Information and Communication Complexity*, pp. 138–149, 2011.

[449] U. M. Richter, *Controlled Self-Organisation Using Learning Classifier Systems.* PhD thesis, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, Karlsruhe, DE, July 2009.

[450] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck, "Observation and Control of Organic Systems," in *Organic Computing - A Paradigm Shift for Complex Systems*, pp. 325 – 338, Birkhäuser Verlag, 2011.

[451] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[452] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[453] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *Journal of Artificial Evolution and Applications*, vol. 2009, p. 1, 2009.

[454] S. Wilson, "Get Real! XCS with Continuous-Valued Inputs," in *Learning Classifier Systems* (P. Lanzi, W. Stolzmann, and S. Wilson, eds.), vol. 1813 of *Lecture Notes in Computer Science*, pp. 209–219, Springer Berlin / Heidelberg, 2000.

[455] W. Lewis, *Tension Structures: Form and Behaviour.* Thomas Telford, 2003.

[456] H.-J. Schock, *Soft shells: design and technology of tensile architecture.* Birkhäuser, 1997.

[457] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler, "Aerial robotic construction towards a new field of architectural research," *International journal of architectural computing*, vol. 10, no. 3, pp. 439–460, 2012.

[458] F. Augugliaro, A. Mirjan, F. Gramazio, M. Kohler, and R. D'Andrea, "Building tensile structures with flying machines," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3487–3492, IEEE, 2013.

[459] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 3, pp. 20–32, 2012.

[460] K. von Frisch, *Animal Architecture*. Harcout Brace Jovanovich, New York, 1974.

[461] M. Hansell, *Animal architecture*. Oxford University Press, 2005.

[462] B. Childers, "Hacking the parrot ar drone," *Linux Journal*, vol. 2014, no. 241, p. 1, 2014.

[463] T. Bohme, U. Schmucker, N. Elkmann, and M. Sack, "Service robots for facade cleaning," in *Industrial Electronics Society, 1998. IECON'98. Proceedings of the 24th Annual Conference of the IEEE*, vol. 2, pp. 1204–1207, IEEE, 1998.

[464] N. Elkmann, T. Felsch, M. Sack, J. Saenz, and J. Hortig, "Innovative service robot systems for facade cleaning of difficult-to-access areas," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1, pp. 756–762, IEEE, 2002.

[465] G. Pérez, L. Rincón, A. Vila, J. M. González, and L. F. Cabeza, "Green vertical systems for buildings as passive systems for energy savings," *Applied energy*, vol. 88, no. 12, pp. 4854–4859, 2011.

[466] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[467] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2149–2154, IEEE, 2004.

[468] M. Hamer, J. Engel, S. Parekh, R. Brindle, and K. Bogert, "ardrone autonomy: a ros driver for parrot ar-drone quadrocopter." Published online `https://github.com/AutonomyLab/ardrone_autonomy`, July 2014.

[469] H. Huang and J. Sturm, "tum simulator." Published online `http://wiki.ros.org/tum_simulator`, July 2014.

[470] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer, "Evolving objects: A general purpose evolutionary computation library," in *Artificial Evolution*, pp. 231–242, Springer, 2002.

[471] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[472] M. J. Wooldridge, *An Introduction to MultiAgent Systems*. West Sussex, UK: John Wiley and Sons Ltd, 2nd ed., 2009.

[473] V. Sarpe, A. Esmaeili, I. Yazdanbod, T. Kubik, M. Richter, and C. Jacob, "Parametric evolution of a bacterial signalling system formalized by membrane computing," in *CEC 2010, IEEE Congress on Evolutionary Computation*, (Barcelona, Spain), pp. 1–8, IEEE Press, 2010.

[474] C. Jacob, V. Sarpe, C. Gingras, and R. Feyt, "Swarm-based simulations for immunobiology," *Information Processing and Biological Systems*, pp. 29–64, 2011.

[475] Unity Technologies, "Unity - game engine." Published online `http://unity3d.com/`, July 2014.

[476] Different Methods, "Swarm agent." Published online `http://u3d.as/content/different-methods/swarm-agent/608`, July 2014.

[477] Allebi Games, "Easy path finding system." Published online `http://u3d.as/content/allebi/easy-path-finding-system/4a7`, July 2014.

[478] Oculus VR, Inc., "Oculus rift: Next gen virtual reality." Published online `http://www.oculusvr.com/rift/`, July 2014.

[479] Razer Inc., "Razer hydra portal 2 bundle." Published online `http://www.razerzone.com/de-de/gaming-controllers/razer-hydra-portal-2-bundle/`, July 2014.

[480] D. Van Krevelen and R. Poelman, "A survey of augmented reality technologies, applications and limitations," *International Journal of Virtual Reality*, vol. 9, no. 2, p. 1, 2010.

[481] L. A. Pegden, T. I. Miles, and G. A. Diaz, "Graphical interpretation of output illustrated by a siman manufacturing system simulation," in *Proceedings of the 17th conference on Winter simulation*, pp. 244–251, ACM, 1985.

[482] R. O'Keefe, "Simulation and expert systems-a taxonomy and some examples," *Simulation*, vol. 46, no. 1, pp. 10–16, 1986.

[483] M. M. Woolfson and G. J. Pert, *An introduction to computer simulation*. Oxford University Press Oxford, 1999.

[484] F. A. Polack, P. S. Andrews, and A. T. Sampson, "The engineering of concurrent simulations of complex systems," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pp. 217–224, IEEE, 2009.

[485] P. Humphreys, "Extending ourselves," *Science at Century's End: Philosophical Questions on the Progress and Limits of Science*, p. 13, 2004.

[486] M. S. McGuire and O. C. Jenkins, *Creating games: Mechanics, content, and technology.* AK Peters Limited, 2009.

[487] T. Susi, M. Johannesson, and P. Backlund, "Serious games: An overview, iki technical reports, hs-iki-tr-07-001," tech. rep., Institutionen för kommunikation och information, 2007.

[488] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović, *et al.*, "Predicting protein structures with a multiplayer online game," *Nature*, vol. 466, no. 7307, pp. 756–760, 2010.

[489] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-time rendering.* CRC Press, 2011.

[490] A. H. Watt and F. Policarpo, *3D games: real-time rendering and software technology*, vol. 1. Addison-Wesley, 2001.

[491] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pp. 281–288, ACM, 2007.

[492] J. Hummel, R. Wolff, T. Stein, A. Gerndt, and T. Kuhlen, "An evaluation of open source physics engines for use in virtual reality assembly simulations," in *Advances in Visual Computing*, pp. 346–357, Springer, 2012.

[493] P. Lindemann, "The gilbert-johnson-keerthi distance algorithm," *Algorithms in Media Informatics*, 2009.

[494] B. Fry, *Visualizing data.* O'Reilly, 2008.

[495] P. C. Wong and R. D. Bergeron, "30 years of multidimensional multivariate visualization.," in *Scientific Visualization*, pp. 3–33, 1994.

[496] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen, "Over two decades of integration-based, geometric flow visualization," in *Computer Graphics Forum*, vol. 29, pp. 1807–1829, Wiley Online Library, 2010.

[497] J. Krause, M. Spicker, L. Wörteler, M. Schäfer, L. Zhang, and H. Strobelt, "Interactive visualization for real-time public transport journey planning," in *SIGRAD*, pp. 95–98, 2012.

[498] W.-K. Jeong, J. Beyer, M. Hadwiger, R. Blue, C. Law, A. Vázquez-Reina, R. C. Reid, J. Lichtman, and H. Pfister, "Ssecrett and neurotrace: interactive visualization and analysis tools for large-scale neuroscience data sets," *Computer Graphics and Applications, IEEE*, vol. 30, no. 3, pp. 58–70, 2010.

[499] B. A. Myers, "A brief history of human-computer interaction technology," *interactions*, vol. 5, no. 2, pp. 44–54, 1998.

[500] A. Dix, *Human computer interaction*. Pearson Education, 2004.

[501] A. Jaimes and N. Sebe, "Multimodal human–computer interaction: A survey," *Computer vision and image understanding*, vol. 108, no. 1, pp. 116–134, 2007.

[502] C. Geiger, R. Fritze, A. Lehmann, and J. Stöcklein, "Hyui: a visual framework for prototyping hybrid user interfaces," in *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pp. 63–70, ACM, 2008.

[503] J. Banks, *Discrete Event System Simulation, 4/e*. Pearson Education India, 2005.

[504] R. E. Nance, "A history of discrete event simulation programming languages," in *History of programming languages—II*, pp. 369–427, ACM, 1996.

[505] C. A. Petri and W. Reisig, "Petri net," *Scholarpedia*, vol. 3, no. 4, p. 6477, 2008.

[506] J. A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, G. Hentschel, S. A. Newman, *et al.*, "Compucell, a multi-model framework for simulation of morphogenesis," *Bioinformatics*, vol. 20, no. 7, pp. 1129–1137, 2004.

[507] A. Wuensche, "Discrete dynamics lab," in *Artificial life models in software*, pp. 215–258, Springer, 2009.

[508] P. Prusinkiewicz, A. Lindenmayer, J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. de Boer, and L. Mercer, *The algorithmic beauty of plants*, vol. 2. Springer-Verlag New York, 1990.

[509] T. Larsson and T. Akenine-Möller, "Collision detection for continuously deforming bodies," in *Proceedings of Eurographics*, pp. 325 – 333, ACM Press, 2001.

[510] T. Larsson and T. Akenine-Möller, "A dynamic bounding volume hierarchy for generalized collision detection," *Computers & Graphics*, vol. 30, no. 3, pp. 450–459, 2006.

[511] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, *et al.*, "Collision detection for deformable objects," in *Computer Graphics Forum*, vol. 24, pp. 61–81, Wiley Online Library, 2005.

[512] A. Husselmann and K. Hawick, "Spatial data structures, sorting and gpu parallelism for situated-agent simulation and visualisation," in *Proc. Int. Conf. on Modelling, Simulation and Visualization Methods (MSV'12)*, pp. 14–20, 2012.

[513] A. R. Stage, N. L. Crookston, and R. A. Monserud, "An aggregation algorithm for increasing the efficiency of population models," *Ecological modelling*, vol. 68, no. 3, pp. 257–271, 1993.

[514] S. Wendel and C. Dibble, "Dynamic agent compression," *Journal of Artificial Societies and Social Simulation*, vol. 10, no. 2, p. 9, 2007.

[515] 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*, vol. 40, pp. 144–160, 2014.

[516] A. S. Shirazi, S. von Mammen, and C. Jacob, "Abstraction of agent interaction processes: Towards large-scale multi-agent models," *Simulation*, vol. 89, no. 4, pp. 524–538, 2013.

[517] I.-K. Hong, J.-B. Ryu, J.-H. Cho, K.-H. Lee, and W.-S. Lee, "Development of a driving simulator for virtual experience and training of drunk driving," in *3rd International Conference on Road Safety and Simulation*, 2011.

[518] J. R. Parker, N. Sorenson, N. Esmaeili, R. Sicre, P. Gil, V. Kochlar, L. Shyba, and J. Heerema, "The booze cruise: Impaired driving in virtual spaces.," *IEEE Computer Graphics and Applications*, vol. 29, no. 2, pp. 6–10, 2009.

[519] T. Schlick, *Molecular Modeling and Simulation: an interdisciplinary guide*, vol. 21 of *Interdisciplinary Applied Mathematics*. Springer-Verlag, New York, 2002.

[520] L. J. S. Allen, *An Introduction to Stochastic Processes with Applications to Biology*. Upper Saddle River, NJ: Pearson Education, 2003.

[521] M. J. M. de Boer and M. de Does, "The relationship between cell division pattern and global shape of young fern gametophytes. i. a model study," *Botanical Gazette*, vol. 151, no. 4, pp. 423–434, 1990.

[522] S. A. Kauffman, *At Home in the Universe: The Search for the Laws of Self-Organization and Complexity*. Oxford University Press, 1995.

[523] G. Paun and G. Rozenberg, "A guide to membrane computing," *Theoretical Computer Science*, vol. 287, pp. 73–100, 9 2002/9/25.

[524] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rosenberg, eds., *Handbook of Graph Grammars and Computing by Fraph Transformation*, vol. 3 of *Concurrency, Parallelism, and Distribution*. Singapore: World Scientific Publishing, 1999.

[525] O. Kniemeyer, G. H. Buck-Sorlin, and W. Kurth, "A graph grammar approach to artificial life," *Artificial Life*, vol. 10, no. 4, pp. 413–431, 2004.

[526] K. Culik and A. Lindenmayer, "Parallel graph generating and graph recurrence systems for multicellular development," *International Journal of General Systems*, vol. 3, no. 1, pp. 53–66, 1976.

[527] M. Nagl, "On the relation between graph grammars and graph l-systems," *Fundamentals of Computation Theory*, pp. 142–151, 1977.

[528] A. Lindenmayer, "An introduction to parallel map generating systems," *Graph-Grammars and Their Application to Computer Science*, pp. 27–40, 1987.

[529] K. Tomita, H. Kurokawa, and S. Murata, "Graph-rewriting automata as a natural extension of cellular automata," *Adaptive Networks*, pp. 291–309, 2009.

[530] H. Sayama and C. Laramee, "Generative Network Automata: A Generalized Framework for Modeling Adaptive Network Dynamics Using Graph Rewritings," *Adaptive Networks*, pp. 311–332, 2009.

[531] S. G. Megason and A. P. McMahon, "A mitogen gradient of dorsal midline wnts organizes growth in the cns," *Development*, vol. 129, pp. 2087–2098, May 2002.

[532] I. R. Edwards and J. K. Aronson, "Adverse drug reactions: definitions, diagnosis, and management," *The Lancet*, vol. 356, no. 9237, pp. 1255–1259, 2000.

[533] 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*, pp. 176–181, Elsevier, 2003.

[534] 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*, vol. 300, no. 20, pp. 2417–2419, 2008.

[535] 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*, vol. 63, no. 6, pp. 919–927, 2014.

[536] M. Eichelbaum, M. Ingelman-Sundberg, and W. E. Evans, "Pharmacogenomics and individualized drug therapy," *Annu. Rev. Med.*, vol. 57, pp. 119–137, 2006.

[537] W. E. Evans and M. V. Relling, "Moving towards individualized medicine with pharmacogenomics," *Nature*, vol. 429, no. 6990, pp. 464–468, 2004.

[538] B. Shastry, "Pharmacogenetics and the concept of individualized medicine," *The Pharmacogenomics Journal*, vol. 6, no. 1, pp. 16–21, 2005.

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

[540] 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*, vol. 422 of *Advances in Intelligent Modelling and Simulation: Artificial Intelligence-based Models and Techniques in Scalable Computing*, pp. 327–349. Springer Verlag, 2012.

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

[542] P. A. Bandettini, "Twenty years of functional mri: the science and the stories," *Neuroimage*, vol. 62, no. 2, pp. 575–588, 2012.

[543] D. Heller, "Combined search in structured and unstructured medical data," in *High-Performance In-Memory Genome Data Analysis*, pp. 181–206, Springer, 2014.

[544] 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*, vol. 8, p. 18, 2013.

[545] 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*, vol. 12, no. 6, pp. 2858–2868, 2013.

[546] N. Gilbert and P. Terna, "How to build and use agent-based models in social science," *Mind & Society*, vol. 1, no. 1, pp. 57–72, 2000.

[547] J. Fisher, D. Harel, and T. A. Henzinger, "Biology as reactivity," *Commun. ACM*, vol. 54, pp. 72–82, Oct. 2011.

[548] R. Axelrod, L. Tesfatsion, and K. L. Judd, *Agent-based Modeling as a Bridge Between Disciplines*, vol. Volume 2, ch. 33, pp. 1565–1584. Elsevier, 2006.

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

[550] J. B. Dabney and T. L. Harman, *Mastering SIMULINK 4*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[551] M. Resnick, S. Ocko, and S. Papert, "Lego, logo, and design.," *Children's Environments Quarterly*, vol. 5, no. 4, pp. 14–18, 1988.

[552] 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*, vol. 52, no. 11, pp. 60–67, 2009.

[553] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *Knowl. Eng. Rev.*, vol. 19, no. 4, pp. 281–316, 2004.

[554] S. Picault and P. Mathieu, "An interaction-oriented model for multi-scale simulation," in *Twenty-Second International Joint Conference on Artificial Intelligence*, (Barcelona, Spain), pp. 332–337, AAAI Press, July 2011.

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

[556] S. von Mammen, D. Phillips, T. Davison, and C. Jacob, "A graph-based developmental swarm representation and algorithm," in *Swarm Intelligence* (M. e. a. Dorigo, ed.), vol. 6234 of *Lecture Notes in Computer Science*, (Brussels, Belgium), pp. 1–12, Springer Verlag, 2010.

[557] J. W. Cooper, "Using design patterns," *Commun. ACM*, vol. 41, pp. 65–68, June 1998.

[558] J. Vlissides and R. Helm, "Pattern hatching: Compounding command," *C++ Report*, pp. 47–52, April 1999.

[559] M. Lam, R. Sethi, J. Ullman, and A. Aho, "Compilers: Principles, techniques, and tools," 2006.

[560] X. Liu, Y. Xiong, and E. A. Lee, "The ptolemy ii framework for visual languages," in *Proceedings of IEEE Symposia on Human-Centric Computing Languages and Environments*, (Stresa, Italy), pp. 50–51, IEEE Press, September 2001.

[561] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller," *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013.

[562] B. E. Perron and A. G. Stearns, "A review of a presentation technology: Prezi," *Research on Social Work Practice*, vol. 21, no. 3, pp. 376–377, 2011.

[563] 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*, vol. 43, no. 1, pp. 79–98, 2001.

[564] C. Reynolds, "Interaction with groups of autonomous characters," in *Game Developers Conference*, pp. 449–460, 2000.

[565] U. Wilensky and CCL at Northwestern University, "Netlogo: A cross-platform multi-agent programmable modeling environment." Published online `http://ccl.northwestern.edu/netlogo/`, March 2014.

[566] P.-P. Grassé, "La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs," *Insectes Sociaux*, vol. 6, no. 1, pp. 41–80, 1959.

[567] M. Dorigo, "Ant colony optimization," *Scholarpedia*, vol. 2, no. 3, p. 1461, 2007.

[568] D. Ebert-May, C. Brewer, and S. Allred, "Innovation in large lectures: Teaching for active learning," *BioScience*, vol. 47, no. 9, pp. 601–607, 1997.

[569] Apple Inc., "Apple bonjour website." Published online `http://www.apple.com/support/bonjour/`, March 2010.

[570] Apple Inc., Cupertino, CA, *Foundation Framework Reference*, 2009.

[571] Apple Inc., "Apple - iphoto." Published online `http://www.apple.com/ilife/iphoto/`, April 2010.

[572] Apple Inc., "Apple - itunes." Published online `http://www.apple.com/itunes/`, April 2010.

[573] Google, "Picasa photo editing." Published online `http://picasa.google.com/`, April 2010.

[574] Nullsoft, "Winamp media player." Published online `http://www.winamp.com/`, April 2010.

[575] E. Hart and P. Ross, "Gavel-a new tool for genetic algorithm visualization," *Evolutionary Computation, IEEE Transactions on*, vol. 5, no. 4, pp. 335–348, 2001.

[576] J. M. Daida, A. M. Hilss, D. J. Ward, and S. L. Long, "Visualizing tree structures in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 6, no. 1, pp. 79–110, 2005.

[577] A. S. Wu, K. A. De Jong, D. S. Burke, J. J. Grefenstette, and C. Loggia Ramsey, "Visual analysis of evolutionary algorithms," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 2, IEEE, 1999.

[578] D. A. Keim and H.-P. Kriegel, "Visdb: Database exploration using multidimensional visualization," *Computer Graphics and Applications, IEEE*, vol. 14, no. 5, pp. 40–49, 1994.

[579] N. Khemka and C. Jacob, "Visplore: a toolkit to explore particle swarms by visual inspection," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 41–48, ACM, 2009.

[580] Apple Inc., "The cocoa framework for mac os x." Published online `http://developer.apple.com/cocoa/`, April 2010.

[581] S. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of theoretical biology*, vol. 22, no. 3, pp. 437–467, 1969.

[582] J. Beal, "Functional blueprints: An approach to modularity in grown systems," in *ANTS 2010: Seventh International Conference on Swarm Intelligence*, pp. 179–190, Springer, 2010.

[583] J. Werfel, "Biologically realistic primitives for engineered morphogenesis," in *ANTS 2010: Seventh International Conference on Swarm Intelligence*, pp. 131–141, Springer, 2010.

[584] I. Salazar-Ciudad and J. Jernvall, "A computational model of teeth and the developmental origins of morphological variation," *Nature*, vol. 464, no. 7288, pp. 583–586, 2010.

[585] S. von Mammen, D. Phillips, T. Davison, H. Jamniczky, B. Hallgrímsson, and C. Jacob, *Morphogenetic Engineering*, ch. Swarm-based Computational Development. Springer Verlag, (in press).

[586] J. Klein, "breve: a 3D environment for the simulation of decentralized systems and artificial life," in *Proceedings of the eighth international conference on Artificial life*, pp. 329–334, MIT Press, 2002.

[587] C. Huepe and M. Aldana, "New tools for characterizing swarming systems: A comparison of minimal models," *Physica A: Statistical Mechanics and its Applications*, vol. 387, no. 12, pp. 2809 – 2822, 2008.

[588] A. Czirok and T. Vicsek, "Collective behavior of interacting self-propelled particles," *Arxiv preprint cond-mat/0611742*, 2006.

[589] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. The Morgan Kaufmann Series in Evolutionary Computation, San Francisco: Morgan Kaufmann Publishers, 2001.

[590] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, no. 1, pp. 29–41, 2002.

[591] M. Pilat, "Wasp-inspired construction algortihms," tech. rep., University of Calgary, 2004.

[592] A. Esmaeili and C. Jacob, "A multi-objective differential evolutionary approach toward more stable gene regulatory networks," *Biosystems*, vol. 98, no. 3, pp. 127–136, 2009.

[593] D. Andre, F. Bennett III, and J. Koza, "Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem," in *Proceedings of the First Annual Conference on Genetic Programming*, (Stanford University, Palo Alto, CA), pp. 3–11, MIT Press, 1996.

[594] C. Lavelle, H. Berry, G. Beslon, F. Ginelli, J.-L. J. Giavitto, Z. Kapoula, A. Le Bivic, N. Peyrieras, O. Radulescu, A. Six, Others, A. L. Bivic, V. Thomas-Vaslin, and

P. Bourgine, "From Molecules to organisms: towards multiscale integrated models of biological systems," *Theoretical Biology Insights*, vol. 1, pp. 13–22, 2008.

[595] S. von Mammen and C. Jacob, "Swarming for games: Immersion in complex systems," in *Applications of Evolutionary Computing, Proceedings Part II*, Lecture Notes in Computer Science, (Tübingen, Germany), pp. pp. 293–302, Springer Verlag, 2009.

[596] S. von Mammen, *Evolving artificial constructive swarms - Experimental models and methodologies*. Saarbrücken, Germany: VDM-Verlag, 2008.

[597] I. Burleigh, "Vigo::3d: A framework for simulating and visualizing of three-dimensional scenes.." Published online `http://vigo.sourceforge.net/docs/`, October 2008.

[598] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani, "An online algorithm for segmenting time series," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, (Washington, DC, USA), pp. 289–296, IEEE Computer Society, 2001.

[599] H. Sakoe, "Dynamic programming algorithm optimization for spoken work recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, pp. 43–49, 1978.

[600] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *AAAI-94 Worshop on Knowledge Discovery in Databases (KDD-94)*, pp. 359–370, 1994.

[601] F. Azuaje, "Computational discrete models of tissue growth and regeneration," *Briefings in Bioinformatics*, vol. 12, no. 1, pp. 64–77, 2011.

[602] J. B. MacQueen, "Some Methods for Classification and Analysis of MultiVariate Observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[603] C.-C. Chen, S. B. Nagl, and C. D. Clack, "Identifying Multi-Level Emergent Behaviors in Agent-Directed Simulations using Complex Event Type Specifications," *Simulation*, vol. 86, pp. 41–51, Oct. 2010.

[604] P. A. Corning, "The re-emergence of "emergence": A venerable concept in search of a theory," *Complexity*, vol. 7, no. 6, pp. 18–30, 2002.

[605] A. Sarraf Shirazi, S. von Mammen, I. Yazdanbod, and C. Jacob, "Self-organized learning of collective behaviours in agent-based simulations," in *ICCS 2011: 8th International Conference on Complex Systems* (H. S. et al., ed.), (Boston, USA), pp. 543–555, NECSI Knowledge Press, 2011.

[606] L. Zhang, Z. Wang, J. a. Sagotsky, and T. S. Deisboeck, "Multiscale agent-based cancer modeling.," *Journal of mathematical biology*, vol. 58, pp. 545–59, Apr. 2009.

[607] S. Bandini, S. Manzoni, and G. Vizzari, "Multi-agent modeling of the immune system: The situated cellular agents approach," *Multiagent and Grid Systems – An International Journal*, vol. 3, pp. 173–182, June 2007.

[608] J.-D. Zucker, "A grounded theory of abstraction in artificial intelligence," *Philosophical transactions of the Royal Society of London Series B, Biological sciences*, vol. 358, pp. 1293–1309, July 2003.

[609] R. L. Goldstone and L. W. Barsalou, "Reuniting perception and conception.," *Cognition*, vol. 65, pp. 231–62, Jan. 1998.

[610] T. Bosse, M. Hoogendoorn, M. C. A. Klein, and J. Treur, "A three-dimensional abstraction framework to compare multi-agent system models," in *Proceedings of the Second international conference on Computational collective intelligence: technologies and applications*, (Berlin, Heidelberg), pp. 306–319, Springer-Verlag, 2010.

[611] T. Ralambondrainy, R. Courdier, and D. Payet, "An Ontology for Observation of Multiagent Based Simulation.," in *IAT Workshops*, pp. 351–354, IEEE Computer Society, 2006.

[612] D. Servat, E. Perrier, J.-P. Treuil, and A. Drogoul, "When Agents Emerge from Agents: Introducing Multi-scale Viewpoints in Multi-agent Simulations," in *Proceedings of the First International Workshop on Multi-Agent Systems and Agent-Based Simulation*, (London, UK), pp. 183–198, Springer-Verlag, 1998.

[613] C.-C. Chen, S. B. Nagl, and C. D. Clack, "Multi-level behaviours in agent-based simulation: colonic crypt cell populations," *Proceedings of the Seventh International Conference on Complex Systems*, 2008.

[614] A. Cardon, "Design and behavior of a massive organization of agents," in *Design of Intelligent Multi-Agent Systems*, vol. 162, pp. 133–190, Springer Berlin / Heidelberg, 2004.

[615] J. Li and M. Kwauk, "Exploring complex systems in chemical engineering–the multi-scale methodology," *Chemical Engineering Science*, vol. 58, no. 3-6, pp. 521–535, 2003.

[616] M. L. Martins, S. C. F. Jr, and M. J. Vilela, "Multiscale models for biological systems," *Current Opinion in Colloid {&} Interface Science*, vol. 15, no. 1-2, pp. 18–23, 2010.

[617] E. Erson and M. Cavusoglu, "A software framework for multiscale and multilevel physiological model integration and simulation," *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, vol. 2008, pp. 5449–5453, 2008.

[618] R. M. H. Merks and J. A. Glazier, "A cell-centered approach to developmental biology," *Physica A: Statistical Mechanics and its Applications*, vol. 352, no. 1, pp. 113–130, 2005.

[619] H. Haken, *Synergetics an Introduction: Nonequilibrium phase transitions and self-organization in physics, chemistry and biology / Hermann Haken*. Springer-Verlag, Berlin, 1977.

[620] J. B. Bassingthwaighte, H. J. Chizeck, and L. E. Atlas, "Strategies and Tactics in Multiscale Modeling of Cell-to-Organ Systems," *Proceedings of the IEEE*, vol. 94, pp. 819–831, Apr. 2006.

[621] H. P. Nii, *Blackboard systems*. STAN-CS, Dept. of Computer Science, Stanford University, 1986.

[622] P. P. González, M. Cárdenas, D. Camacho, A. Franyuti, O. Rosas, and J. Lagúnez-Otero, "Cellulat: an agent-based intracellular signalling model," *Bio Systems*, vol. 68, no. 2-3, pp. 171–85, 2003.

[623] S. Khan, R. Makkena, F. McGeary, K. Decker, W. Gillis, and C. Schmidt, "A multi-agent system for the quantitative simulation of biological networks," in *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, (New York, NY, USA), pp. 385–392, ACM, 2003.

[624] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: Macmillan, 1994.

[625] Z. Wang, C. M. Birch, J. Sagotsky, and T. S. Deisboeck, "Cross-scale, cross-pathway evaluation using an agent-based non-small cell lung cancer model.," *Bioinformatics (Oxford, England)*, vol. 25, pp. 2389–96, Sept. 2009.

[626] M. Scheutz and P. Schermerhorn, "Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models," *Journal of Parallel and Distributed Computing*, vol. 66, no. 8, pp. 1037–1051, 2006.

[627] M. Lysenko and R. M. D'Souza, "A framework for megascale agent based model simulations on graphics processing units," *Journal of Artificial Societies and Social Simulation*, vol. 11, no. 4, p. 10, 2008.

[628] S. von Mammen, A. Sarraf Shirazi, V. Sarpe, and C. Jacob, "Optimization of swarm-based simulations," *ISRN Artificial Intelligence*, vol. Article ID 365791, pp. 1–12, 2012.

[629] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "GPGPU: general-purpose computation on graphics hardware," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, (New York, NY, USA), p. 208, ACM, 2006.

[630] J.-P. Müller, "Emergence of Collective Behaviour and Problem Solving," in *Engineering Societies in the Agents World IV* (A. Omicini, P. Petta, and J. Pitt, eds.), vol. 3071 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2004.

[631] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, vol. 55, no. 3, p. 601, 1983.

[632] A. Deutsch and S. Dormann, "Introduction and Outline," in *Cellular Automaton Modeling of Biological Pattern Formation*, Modeling and Simulation in Science, Engineering and Technology, pp. 3–11, Birkhäuser Boston, 2005.

[633] M. Gardner, "Mathematical games: The fantastic combinations of john conway's new solitaire game "life"," *Scientific American*, vol. 223, pp. 120–123, October 1970.

[634] A. Deutsch and S. Dormann, *Cellular Automaton Modeling of Biological Pattern Formation*. Birkhäuser Boston, 2005.

[635] X.-S. Yang and Y. Young, "Cellular Automata, PDEs, and Pattern Formation," *Handbook of Bioinspired Algorithms and Applications*, p. 12, Mar. 2010.

[636] R. M. Amorim, R. S. Campos, M. Lobosco, C. Jacob, and R. W. dos Santos, "An Electro-Mechanical Cardiac Simulator Based on Cellular Automata and Mass-Spring Models," in *ACRI* (G. C. Sirakoulis and S. Bandini, eds.), vol. 7495 of *Lecture Notes in Computer Science*, pp. 434–443, Springer, 2012.

[637] E. Bonabeau, "From classical models of morphogenesis to agent-based models of pattern formation.," *Artificial life*, vol. 3, pp. 191–211, Jan. 1997.

[638] G. Vizzari and L. Manenti, "An agent-based model for pedestrian and group dynamics: experimental and real-world scenarios," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, (Richland, SC), pp. 1341–1342, International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[639] M. Batty, "Agent-based pedestrian modelling," *Advanced spatial analysis: the CASA book of GIS*, p. 81, 2003.

[640] T. C. Schelling, "Dynamic Models of Segregation," *Journal of Mathematical Sociology*, vol. 1, pp. 143–186, 1971.

[641] M. Abdou, N. Gilbert, and K. Tyler, "Agent-Based Simulation Model for Social and Workplace Segregation," in *Proceedings of the 8 Annual Conference of European Social Simulation Assoc.*, (Brescia), pp. 1–12, 2008.

[642] B. Dahlbäck, "Blood coagulation," *Lancet*, vol. 355, pp. 1627–32, May 2000.

[643] H. Haken, "Synergetics," *Naturwissenschaften*, no. 3, pp. 121–128, 1980.

[644] A. Fuchs, *Self-organization and Synergetics*, pp. 147–157. Springer Berlin Heidelberg, 2013.

[645] T. Möller, E. Haines, and N. Hoffman, *Real-time rendering*. AK Peters Ltd, 2008.

[646] E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces," *Artificial intelligence*, vol. 5, no. 2, pp. 115–135, 1974.

[647] E. H. Durfee, "Practically coordinating," *AI Magazine*, vol. 20, no. 1, pp. 99–116, 1999.

[648] S. Ramchurn, N. Jennings, C. Sierra, and L. Godo, "Devising a trust model for multi-agent interactions using confidence and reputation," *Applied Artificial Intelligence*, vol. 18, no. 9, pp. 833–852, 2004.

[649] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *Proceedings of the International Conference on Data Engineering*, pp. 106–115, Citeseer, 1999.

[650] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 493–498, ACM, 2003.

[651] E. Fuchs, T. Gruber, J. Nitschke, and B. Sick, "On-line motif detection in time series with swiftmotif," *Pattern Recognition*, vol. 42, no. 11, pp. 3015 – 3031, 2009.

[652] Q. Wang, V. Megalooikonomou, and C. Faloutsos, "Time series analysis with multiple resolutions," *Information Systems*, vol. 35, no. 1, pp. 56 – 74, 2010.

[653] S. von Mammen, S. Schellmoser, C. Jacob, and J. Hähner, *The Digital Patient: Advancing Medical Research, Education, and Practice*, ch. 11. Modelling & Understanding the Human Body with Swarmscript, pp. 149–170. Wiley, 2016.

[654] J. Kiefer, "Conditional confidence statements and confidence estimators," *Journal of the American Statistical Association*, vol. 72, no. 360, pp. 789–808, 1977.

[655] T. A. Louis and S. L. Zeger, "Effective communication of standard errors and confidence intervals," *Biostatistics*, vol. 10, no. 1, p. 1, 2009.

[656] R. Kiefhaber, G. Anders, F. Siefert, T. Ungerer, and W. Reif, "Confidence as a means to assess the accuracy of trust values," in *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, (Washington, DC, USA), pp. 690–697, IEEE Computer Society, 2012.

[657] H. Prothmann, S. Tomforde, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, "Organic traffic control," in *Organic Computing — A Paradigm Shift for Complex Systems* (C. Müller-Schloer, H. Schmeck, and T. Ungerer, eds.), vol. 1 of *Autonomic Systems*, pp. 431–446, Springer Basel, 2011.

[658] P. Eugster, P. Felber, and F. Le Fessant, "The "art" of programming gossip-based systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 37–42, Oct. 2007.

[659] J.-P. Steghöfer, P. Behrmann, G. Anders, F. Siefert, and W. Reif, "Hispada: Self-organising hierarchies for large-scale multi-agent systems," in *ICAS 2013, The Ninth International Conference on Autonomic and Autonomous Systems*, pp. 71–76, 2013.

[660] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[661] H. Lipson and J. B. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature*, vol. 406, pp. 974–978, August 31 2000.

[662] M. D. Schmidt and H. Lipson, "Actively probing and modeling users in interactive co-evolution," in *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, (New York, NY, USA), pp. 385–386, ACM Press, 2006.

[663] J. B. Predd, S. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *Signal Processing Magazine, IEEE*, vol. 23, no. 4, pp. 56–69, 2006.