



UvA-DARE (Digital Academic Repository)

Modelling with cellular automata: problem solving environments and multidimensional applications

Naumov, L.A.

Publication date
2011

[Link to publication](#)

Citation for published version (APA):

Naumov, L. A. (2011). *Modelling with cellular automata: problem solving environments and multidimensional applications*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

Chapter 1. Introduction

1.1 The Third Paradigm

Half a century ago it was appropriate to distinguish two significantly different, but tightly interconnected and continuously interacting research paradigms: “an experiment” and “a theory”. An experiment (from Latin “experiri” – “to try”) is an empirical way of cognition. It generally consists in reproducing some aspects of natural phenomena in a well-studied and controllable artificial environment. Basing on experimental results scientists are able to quantify phenomena and formulate, prove or reject hypotheses. Observations can be generalized and assembled into a theory. A theory in general is the highest and mostly advanced form of the scientific knowledge organization notwithstanding the fact that Einstein said: “Nobody who has really gone deeply into the matter will deny that in practice the world of phenomena uniquely determines the theoretical system, in spite of the fact that there is no logical bridge between phenomena and their theoretical principles” [1].

With the advent of the computer as a research tool in the middle of the previous century, a third research paradigm: “simulation” or “modelling” entered the realm of science. According to this paradigm the behavior of the studied system is being reproduced by means of computations.

The aim of both experimentation and simulation consists in reproducing some aspects of nature, but there is a significant difference. Experimental research uses nature as a processing vehicle. An experiment mainly consists in preparation of initial setup, supporting necessary conditions, and analyzing results. An in vivo or in vitro process is led and controlled by nature itself. At the same time a model is to reproduce the phenomenon in a totally artificial way, imitating not only the object of study, but also all the required laws of nature.

Motivations and roles are also different: experiments can be performed on any stages of cognition. They are applicable for the initial study of the phenomenon, when simulations can be started only after a detailed research has been performed and a significant knowledge has been accumulated.

Both paradigms allow to verify hypothesis and get data, but modelling gives an additional opportunity to check the level of cognition. Robert M. Pirsig has written that “the real purpose of scientific method is to make sure nature hasn’t misled you into thinking you know something you don’t actually know” [2]. A simulation is one of the methods to assure.

Explanation is the way for validating the level of cognition. If one will be able to describe given aspect of nature’s behavior as an algorithm for the machine and

the validation confirms that the model reproduces it in a good agreement, this means that a distinct level of cognition has been reached. That is why the appearance of the simulation paradigm has not devaluated the experimental approach, to the contrary, the importance of both have risen due to their complementarity, mutual validation, and comparison opportunities.

At the same time *in silico* modelling of a phenomenon has advantages over *in vitro* or *in vivo* experimenting. They are listed below.

- Modelling allows relatively low cost and fast test of hypothesis.
- Simulation can be performed even if an experiment is not possible at all, for example, dealing with materials, which are not accessible in the laboratory or even not yet invented.
- Often modelling allows to predict or achieve results faster than the experiment does. This gives an opportunity to outrun nature. Such knowledge, which will be obtained in advance, can play significant role in tuning an *in vitro* setup or modifying an *in vivo* system. An important application of this ability lies, for example, in the area of biomedical simulations, where such benefits can save lives (see Chap. 4).
- *In silico* modelling allows to use a rich set of analysis facilities. Computers give an opportunity to retrieve or variate any possible characteristics of the data set, which represents the phenomenon. These characteristics can even be hardly observable *in vivo* or *in vitro* (see Sect. 4.4.5).

A common process of modelling is schematically represented in Fig. 1.1. Four principal phases of the process are distinguished.

Following list explains this workflow using the example of a fluid flow simulation.

1. The **phenomenon** to be modelled is the flow of some fluid.
2. After **abstracting** from all unnecessary properties of a fluid, there will be at least two variants for the **conceptual model**:
 - Represent liquid as a continuous field of velocities.
 - Look at liquid as volume packets.
3. The choice of first variant above will lead to the description of a **computational model** using Navier-Stokes equations [3]. The second variant brings the Lattice Boltzmann equation [4]. Despite the fact that the application areas of these two computational models can be completely different, they are interchangeable on the level of concepts or implementation. Finally, if it is assumed that time, space and state are discrete then a computational model can be **designed** as a cellular automaton.
4. As soon as the automaton will be **implemented**, **computations** can be started. This will lead to some result and the question of its **validation** will bring us back to the observation of the fluid flow.

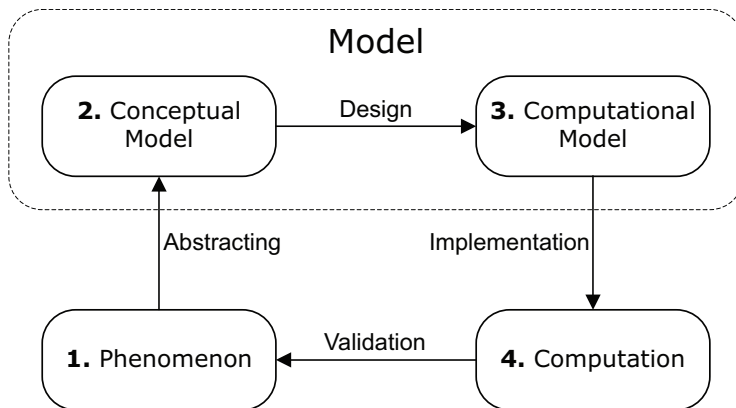


Figure 1.1: Schematic representation of the general modelling workflow. Four principal phases of the process are distinguished and represented as nodes on the graph. Actions, which lead from one phase to another, are written on the edges. Conceptual and computational models together form a model of a phenomenon.

1.2 Problem Solving Environments for Simulations

Modern computational problems are multidisciplinary and require knowledge from different domains. Nowadays researchers need to share experience, methodologies, data and resources to be able to approach their challenging tasks. This provokes the appearance of a special kind of facilities, so called “problem solving environments” [5–8]. This term refers to the “computer system that provides all computational facility necessary to solve a target class of problems” [5]. Having such a general definition, possible approaches to the creation of problem solving environments are intensively discussed by the scientific community.

The main thing, which a problem solving environment should offer to a researcher, is the ability to describe, solve and perceive the solution using the language of the problem. At the same time a problem solving environment should use maximum of possible benefits of the computational architecture, the operating system and whatever it can, hiding all the complexities, related to this.

The amount of scientific knowledge about natural phenomena is permanently growing, reaching levels which would be considered as unimaginable several decades ago. For sure, this knowledge can be successfully utilized for simulations in different areas of science. However this is hampered by the fact that the complexity of modern computational instruments for simulations requires special education of a researcher.

Applied computational science is one of the most productive fields of science nowadays. This fact is tightly connected with the creation of myriads of specialized problem solving environments. Making the process of modelling more comprehensible, they represent facilities to reach a next level of the phenomena’s cognition.

Problem solving environments, designed for the simulation of a distinct class of phenomena can be named as modelling environments. General and complete mod-

elling environments should support the whole process depicted in Fig. 1.1 covering all phases of the modelling workflow. Obviously, the phenomenon itself stays out of the modeling facility, but such tool can still be of use for collecting an in vivo or in vitro experimental data.

Table 1.1 presents an example of the schematic communication between a researcher and an idealized complete modelling environment to illustrate both aims and benefits of such facilities. The second column contains comments for the actions, which should be undertaken during the distinct phase of the workflow, shown in the first column. The third column explains what kind of functionality is required from the environment for the given phase.

<i>Phase</i>	<i>Comments</i>	<i>Expectations</i>
1. Phenomenon	First of all, a researcher should study the phenomenon. This could be done in vivo, in vitro, or by reading literature depending on the final aim and current possibilities.	An environment can help to observe the phenomenon, participating even on the initial stages of studying. For example, it can provide functionality for making snapshots (photographs or any other appropriate) of the experiment's state or collect data from sensors or other involved devices.
2. Conceptual Model	After the basic study of the phenomenon has been accomplished a researcher should distinguish significant properties, which are to be simulated. Then the laws, which describe an evolution of these properties, are to be formulated and formalized. Usually such laws will be represented with formulae, equations and functions.	An environment can provide a language of mathematical abstractions for dealing with properties and corresponding laws descriptions (formulae, equations and functions), which at the same time should be close to the language of the target problem domain, mimicking common mathematics into abstractions from the area of study. The environment can also support mathematical transformations, solving and analyzing capabilities. The perspective of the extent in this direction is very rich [9].

<h3>3. Computational Model</h3>	<p>The conceptual model should be converted to a form appropriate for modelling, which, among the other, depends on answers onto three questions [10]:</p> <ol style="list-style-type: none"> 1. Should time be continuous or discrete? 2. Should space be continuous or discrete? 3. Should a set of possible states be continuous or finite? <p>If all of them are decided to be continuous then the model can be described in terms of partial-differential equations. If time is the only discrete measure then integro-differential equations should be used. If only space is discrete then coupled ordinary differential equations will fit. If state and space are discrete, but only time is continuous then interacting particle systems should be used. If only state is continuous then coupled map lattices or systems of differential equations can be appropriate. Models where state, space and time are discrete belong to the area of cellular automata.</p>	<p>An environment should provide a language for dealing with computational models, described using an appropriate formalism. It should allow to define initial and boundary conditions. One of the most important differences between conceptual and computational models is that the conceptual model is attempting to describe the phenomenon, whereas a computational model is describing the conceptual one, taking the area of interest and the specifics of the simulation process into account. So, a computational model belongs to a higher level of abstraction, being farther from the phenomenon. So parameters, which do not relate to the phenomenon, but reflect and describe the specifics of the phenomenon's in silico representation and the process of relevant computations belongs to computational model.</p>
---------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4. Computation	The computational model should be executed on a given computational architecture with the help of the appropriate solver.	Traditionally, this is the main area, where modelling environments can show all their benefits and power. It should allow to tune the computational experiment and perform it, using modern methods, appropriate for distinct classes of models. It can also optimize, analyze and visualize the process, store states of experiments for postponing computations and sharing results with other researchers. Finally it can support validation capabilities, which involve data, retrieved on the first phase of the modelling workflow.
----------------	---------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 1.1: Example of the schematic communication between an idealized and complete modelling environment and its user, a researcher. The first column contains a name the workflow phase, the second column describes actions, and the third column presents what can be expected from an idealized environment.

As was mentioned above, the forth phase of the modelling process is the main area, where not idealized, but a real modelling environment acts. When speaking about computations one of the key characteristic of the process is its productivity or effectiveness. The characteristics, which could be opposed to the productivity of the facility is its universality or generality. Usually more universal tool demonstrates inferior productivity than a specialized one. But on the other hand the power of computers grows exponentially. As a result, the modern trend in software engineering consists in creation of universal tools, despite of their system requirements. Such tools often cannot even run on computers, which were topical half a decade ago, but nowadays this is compensated by the progress of modern hardware.

There is an enormous increase in both the speed of computers and in the algorithmic power [11]. History has a helical character and such situation repeats the history of programming languages development. First widely used programming languages were members of the Assembly family. These low-level languages are close to the machine code, so they are far from being universal, reflecting the specifics, when using on a given processor.

From 50s till early 70s the “programmer” was the member of a special closed cast, because usage of such languages required special education, which cannot be substituted by reading a book. The situation has changed when C [12] and similar programming languages of relatively higher level appeared. They were a bit more universal, being posed not for distinct processors, but for processor families. Creation of C++ [13] in 1983 was one of the most important events in the history of programing, but Assemblers were still in use and were considered to be languages, which allow to

produce most effective* code.

The situation has changed with the progress of computers regardless the progress of languages. As soon as the complexity of CPUs with multiple pipes reached a distinct level it became obvious that optimizing compilers for higher level languages will be able to prepare much more effective code than an Assembler programmer. Even from that time the productiveness of computers jumped ahead and universal, cross-platform languages appeared on the scene. Contemporary processors have multiple processing units, each of which has multiple cores with multiple pipes. Nowadays, even such computational tools as Waterloo Maple [9], Globus Toolkit [14], or Ibis [15], for which productivity really matters, are implemented in Java [16]. Cross-platformability has a priority over the fact that its code is interpreted and “just-in-time” compiled.

The universality in terms of problem solving environments does not mean its portability or cross-platformability, but refers to the extent of a class of problems it can handle. Design of an environment always includes the consideration of a compromise between the universality and the price.

1.3 Cellular Automata for Simulations

Cellular automata are discrete dynamic systems, which can be fully described in terms of local interactions [10, 17, 18]. The cellular automata theory was invented half a century ago. The exact author of this field cannot be named definitely. In 1948 John von Neumann [19] gave a lecture entitled “The General and Logical Theory of Automata”, where he presented his ideas of universal and self-reproducing machines. According to his own statement, his work was inspired by Stanislaw Ulam [20]. Konrad Zuse [21] also suggested that the universe could be a cellular automaton and used this idea for development of the computing machines. At the same time, some members of the scientific society regard the paper by Wiener and Rosenblueth [22], or the mathematical work that was done in early 1930s in Russia as the start of the field [10].

As was said in Table 1.1, cellular automata present a general computational model with discrete time, space and state. One of the most important causes of interest to these systems is that they form a common paradigm of parallel computations as Turing machines [23, 24] do for the consecutive computations [17]. Taking their natural parallelism into account, cellular automata can be utilized to model large scale emergent systems on parallel computers. Cellular modelling facilities provide useful and demanded tools for programming emergent computations that model complex phenomena in many application domains from science and engineering to economics and social sciences [25]. Such automata can represent a universal problem specification language for arbitrary tasks of parallel computing and, especially, for simulations of spatial-distributed (or distributed over other parameters) systems. That is why cellular automata are widely used as computational vehicles for modelling environments.

From a formal point of view a cellular automaton A can be defined as a set of four objects $\{G, Z, N, f\}$ [26], where:

*Here we mean the effectiveness in terms of execution speed. So, the “most effective” means “fastest”.

- G – *the grid*, a finite metric space of cells. A term “grid’s state” designates the aggregate of all cells’ states[†].
- Z – *finite set of possible cells states*. Often cell’s state is to be characterized by floating-point values. It is possible, because such values have not a continuous, but a finite codomain in silico [27, 28].
- N – *the description of cell’s neighbourhood*. The neighbourhood is the set of cells which have an influence to the considered one. In most cases the neighbourhood is described in term of relative offsets or the acceptable range of distances. More complex cases like Margolus neighbourhood [17] need special kind of definition. Nevertheless, in most cases the neighbourhood of a cell is a subset of cell’s nearest neighbors. The exact set of actual neighbors of the cell c , which can be found according to description N will be denoted as $N(c)$.
- f – *the transition function* which determines the next state for each cell depending on its neighbourhood. There are two variants for function operating: $Z^{|N|+1} \rightarrow Z$ and $Z^{|N|} \rightarrow Z$. The first variant corresponds to the situation if automaton has cells “with memory”. In such case the transition function accounts the current state of the cell, which it is applied to. The second variant correspond to a function of an automaton with “memoryless” cells. The difference between these two types of transition functions isn’t important in practice, but has a theoretical value [29]. For real applications a transition function can be defined in a more general way, as a computable program on some programming language (probably, a specialized one [17, 30–32]) instead of the mathematical and logical expressions. Transition function can be also referred as “*rules*” or “*laws*” of a cellular automaton.

Classical cellular automata (they are also called as “synchronous cellular automata”) are characterized by the following basic properties.

- *Locality* – automaton’s rules should be local. All members of cell’s neighbourhood are to be on a finite distance from the cell.
- *Homogeneity* – the system should be similar for all the cells. No regions of the grid should be distinguishable by the landscape.
- *Synchronicity* – all cells accept their new states simultaneously, at the end of the iteration, after new values were computed for all of them. In other words, new state of the grid should not depend on the order of cells traversal.

At the same time there is number of classical cellular automata modifications, which neglect some of these properties or add other features to make the automaton more suitable for the distinct spectrum of tasks. Widely used non-classical cellular automata and their key distinctions are presented in the following list.

[†]Here and further the statement “the cell is in state s ” and “cell contains/stores value s ” is assumed to be synonymous.

- *Asynchronous* automata neglect the synchronicity requirement. This makes the grid's state dependent on the order of the cells updating, but such fact can be useful for some tasks. Such automata requires a factor of two less memory for the grid's state storage. As a result it can provide better throughput. In addition asynchronous automata allow faster signal[‡] transportation over the grid in the direction of the cells traversal [33].
- *Heterogeneous* automata neglect homogeneity requirement, allowing, for example, local grid refinement.
- There are two ways of determining *stochastic* or *probabilistic* cellular automata. According to the first one, cells of stochastic automaton change their states with distinct set of probabilities. From a formal point of view, transition function f is substituted by set of k transition functions $\{f_i\}_{i=1}^k$, which are chosen for any given cell with distinct probability each. This choice is controlled by a set of probabilities $\{p_i\}_{i=1}^k$, where $\sum_{i=1}^k p_i = 1$.

According to the second way of determining, stochastic cellular automaton represents a classical cellular automaton, which update only randomly chosen cells on each timestep. Obviously, this definition is a particular case of the previous one. Such automata are widely used nowadays [34, 35].

There are a lot of more specific cellular automata modifications for distinct purposes, like lattice gas automata [25, 36] for fluid dynamics, movable cellular automata [37, 38] for computational solid mechanics, Brownian cellular automata [39] for signal propagation simulation and many others [40, 41]. In any case, a “cellular automaton” refers to a complex system which has a distinct set of properties and internal notions: grid, cell, cell's state, neighbourhood, rules etc. When a modelling environment uses the language of cellular automata as a basic abstraction for the computational model formulation, this means that properties and notions mentioned above are to be supported and offered to a researcher.

A classical cellular automaton (and most of its modifications) is a homogeneous parallel architecture. At the same time, a modelling environment can give an opportunity to execute a simulation on cluster [42] or Grid systems [43], which often are heterogeneous. An environment can also allow an execution on a standalone computer, which is not parallel in massive sense[§]. In this case the question is: what is “modelled” by the modelling environment? The primary object of modelling is the cellular automaton with all its features in given computational conditions. Secondary object of modelling is a phenomenon, which is reproduced in the framework of the cellular automaton.

Further discussion will cover only modelling environments, which use cellular automata as the computational vehicle.

[‡]The information about the state of previously considered cells is meant as “signal” here.

[§]At the same time, it can be parallel, being a multicore machine, for example.

1.4 Thesis Outline

The outline of the thesis is as follows:

Chapter 2 answers the question: what are the invariant requirements for a general cellular automata based modelling environment? This is followed by an overview of existing tools of this class. Their conformance to the list of the requirements is analyzed and compared. Despite the fact that a lot of such tools appeared in past decade, we motivate the need of a new one and describe its key ideas and advantages. This resulted in “CAME_{&L}” the “Cellular Automata Modelling Environment & Library”.

Chapter 3 presents an application of CAME_{&L} for theoretical studies. We perform a classification of structures generated by one-dimensional binary cellular automata from a single seed. The question we want to answer is how many and what kind of classes of equivalence can be distinguished for these structures? What is the number of really different shapes of these structures? Is it possible for a structure generated by the automaton with cells, which have memory, to belong to the same class as a structure, generated by the automaton with memoryless cells? The second part is devoted to a formalism of generalized coordinates, which presents a universal method of state data indexing for different cellular automata grids. We propose a set of different methods of such coordinates for all the possible two-dimensional grids, composed of regular polygons.

Chapter 4 demonstrates an application of CAME_{&L} in the area of tumour modelling. Here we present two three-dimensional cellular automata based models of tumour growth. With the help of these models we study the relevance of the tumour natural shrinkage and its influence on the growth regime. In addition we introduce several algorithmic substitutions to optimise computations. The scientific question, we are going to answer here, is following: does these tricks influence the realism of in silico tumour growth? Moreover, involving one of our models, we study the influence of the successful mitoses rate on the tumour dynamics.

Chapter 5 summarizes the thesis with a discussion and conclusions.