



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 2. Problem Solving Environment for Cellular Automata Based Simulations

2.1 Features and Requirements

In the past decade a lot of cellular automata environments have been implemented on current desktop computers [17, 31, 44–59]. Simulation of large scale two or three dimensional cellular automata results in an enormous computational load. There are two main alternatives that allow to achieve needed performance in the implementation of cellular automata. The first one consists in the design of special hardware devoted to the execution of cellular automata. The second alternative is based on the use of the commercially-available parallel computers for developing parallel cellular automata applications and environments.

As explained in Sect. 1.3 cellular automata modelling software and hardware systems belong to the class of problem solving environments. The community has formulated the following common recommendations for a general problem solving environment:

1. It should **reduce the difficulty** of the simulation [7].
2. It should **reduce costs and time** of complex solutions development [7].
3. It should allow to perform experiments **reliably** [7].
4. It should have a **long lifetime** without getting obsolete [7].
5. It should **support the plug-and-play paradigm** [7].
6. It should **exploit** the paradigm of the **multilevel abstractions** and complex properties of science [7].
7. User should be able to **use the environment without any specialized knowledge** of the underlying computer hardware or software [5].

This chapter is partly based on: D. Talia, L. Naumov. Parallel Cellular Programming for Emergent Computation. In A. Hoekstra, J. Kroc, P. Sloot (Eds.) *Simulating Complex Systems by Cellular Automata*. pp. 357–384. Springer. 2010. It is also based on: L. Naumov. *CAME_gL – Cellular Automata Modelling Environment & Library*. Cellular Automata: 6th International Conference on Cellular Automata for Research and Industry (ACRI-2004). pp 735–744. Springer-Verlag. 2004.

8. It should be addressed to the **wide scope** of problems [5].
9. It should be able to coordinate **large computational power** [5].
10. It should be **complete**, containing and providing all facilities, which might be required for solving a problem in a target domain [6].
11. **Extensibility** of the environment will provide an ability to enlarge the target problem domain and to enrich the set of supported functions. This can be achieved with the help of a component-based design. A component approach also complies with the modern trend in distributed problem-solving facilities design, which is based on web- and grid services [7, 14, 60] or Common Object Request Broker Architecture (CORBA) objects [8, 61].

Basing on common considerations, the software or hardware modelling facility, which allows to perform experiments using cellular automata, should have the following set of features:

1. It should **hide the complexity** of a computational architecture, operating system, networking mechanism and other technical aspects, which are far from a research domain. The language which should be used to control the environment has to use abstractions, which correspond to basic cellular automata concepts or to the target problem field.
2. It should allow to **setup and tune** a cellular automaton for the computational experiment, granting the researcher as much freedom as it is possible.
3. It should provide tools to **run and control** a computational experiment. A good solver should use maximum of benefits provided by the computational architecture on one hand and by cellular automaton rules on another hand.
4. It should support **visualization** of the grid's state and other experimental data. This feature plays one of the primary roles in understanding the phenomenon, especially, when modelling spatially-distributed systems.
5. It should provide a set of tools to **analyze** grid's state or any computational experiment's characteristics, which is possible to obtain.
6. It should provide **reproducibility** and allow to share experiments description between researchers.

The following list consists of concrete features of a general cellular automata based problem solving environment. This list is based on two previous lists, so at the end of each entry there are references to recommendations or requirements introduced above, given in italics. References to the first list are preceded by "*PSE:*", whereas references to the second one are preceded by "*CA:*". Moreover, the "Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science" [5, 62]

produced “Findings” and “Recommendations” for problem solving environments. We will not cite them here, but they will be referenced in the “*PSE:*” section as “ F_n ” or “ R_n ” respectively, where n is the number of the specific finding or recommendation stated in [5].

1. The environment should be as **universal and customizable** as possible. This property is interpreted in the sense of a spectrum of problems it could be useful for and should not be understood as “cross-platformability”. Support of a variety of grids, possible states, neighbourhoods and boundary conditions is desirable or even necessary. The environment should allow to choose and modify the type of the automaton to be modeled and parameters of the experiment from the widest possible spectrum of variants (*PSE: 2, 8, 10; CA: 2*).
2. **Extensibility** is a contemporary and actively used property of software and hardware systems and consists in the ability of seamless incorporation of novel functionality or algorithms (*PSE: 2, 4, 5, 6, 8, 9, 10, 11, F₆, R₁; CA: 1*).
3. The environment should support **modern parallel or distributed computational technologies**. For a software cellular automata system this means that it should involve multicore, cluster [42] or Grid [43] computing, Message Passing Interface (MPI) [63], Parallel Virtual Machine (PVM) [64], OpenMP [65] or other technologies, because the software has to emulate a homogeneous parallel architecture of cellular automaton, but not counterfeit it (*PSE: 1, 2, 3, 9, 10, F₁, F₂, F₃; CA: 1, 3*).
4. A solver can include **optimization** capabilities, to improve the performance of computations by rearranging operations, excluding constant areas of the grid, which will not change on closest steps and so on (*PSE: F₁, F₂, F₃; CA: 3*).
5. The environment should provide visually attractive, handy and clear **user interface**. It has to preserve the interactivity even when performing long experiments, providing a reliable control (*PSE: 1, 2, 7, F₂, F₄, F₇, R₇; CA: 1, 2, 3*).
6. As was already said, an experiment **description language** has to be close to the target problem domain and as far from the implementation details as possible. Description itself should be independent of the computational architecture, available resources, and operating system. The ability to involve such dependencies should be definitely considered as a powerful option (*PSE: 1, 2, 6, 7, 10, F₁, F₄, R₇; CA: 1, 2*).
7. **Visualization** of the grid’s state is the most straightforward way of the experiment visual representation. Nowadays scientific visualization seems to be a separate industry [66]. So there is no need for a cellular automata based problem solving environment to be competitive with top-level tools in this area. Nevertheless the environment should contain a basic set of features and, probably, be compatible with specialized visualization software on the level of data-files (*PSE: 2, 10, R₇; CA: 4*).

8. The environment should support **analysis** functionality to monitor the quality of the experiment, study its progress and final results (*PSE: 2, 6, 10, R₇; CA: 5*).
9. The environment should allow to **reproduce** experiment on the same or another computational system. This will give an opportunity to share knowledge and experience between researchers, eliminate repetitive computations, postpone the simulation, reanalyze, revisualize or generally reuse data. Such abilities can be achieved by support of storing/restoring full computational experiment setup and grid's state to/from a file (*PSE: 1, 2, 4, 10, F₁, F₄, R₆; CA: 6*).

2.2 Survey of Existing Cellular Automata Based Problem Solving Environments

The amount of software environments created for cellular automata modelling is impressive [32, 48]. The apogee of this boom was in 1990s. The majority of projects made that time has been already outdated, but some new successful prototypes appeared.

A comparative survey of existing software and hardware facilities is summarized in Table 2.1. The first column contains the name of the project with references. The second column presents the target platform and the third contains the year of the latest known release (or the last publication devoted to the tool). Next nine columns contain pluses or minuses, expressing if the project satisfies the requirements listed in Sect. 2.1. The analysis, which allows to make a conclusion about the conformance of a tool with this or that requirement, was performed by running them or studying the referred publications. To be more objective, Boolean marks were chosen. The tool gets positive mark even if there was any step made in the direction of the requirement. It was decided to estimate facilities regardless a scale of this step, because the scale can be determined unfairly.

It is impossible and useless to overview all the existing projects, which were created ever. Those of them which last known version had been released in XX century were mostly excluded from the study. Only several of tools created that time are listed, because they have significant historical value. At the same time, some relatively new projects are deliberately not presented if they are deemed to be unsuitable for the research and scientific modelling.

Projects are listed in the alphabetical order. In the following there are short, one-paragraph reviews, which briefly describe each project in more detail. The name of the project which is subjected to the review is shown with bold when it appears for the first time.

CAGE, the “Cellular Automata General Environment” does not support any parallel or distributed computational technology, which is compensated by its universality. The authors have generalized the notion of the “cellular automaton” and used this vision for the software design. The environment supports multilayered grids, rich means for the neighbourhoods’ formation including a query-based one. It supports

<i>Name [references]</i>	<i>Platform</i>	<i>Update</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
CAGE [44]	Windows	2004	+	-	-	-	+	+	+	-	+
CAM [17, 45]	iX86 or Sun	1994	+	+	+	-	+	+	+	-	+
CAMEL [31, 46, 47]	UNIX/Linux	1996	+	-	+	-	+	+	+	-	+
CAME _{&} L [48- 50]	Windows	2010	+	+	+	+	+	+	+	+	+
CAMELot [51]	UNIX/Linux	2001	+	-	+	-	+	+	+	+	+
Cellular [52]	UNIX/Linux or Windows (CygWin)	1999	+	+	-	-	+	+	+	-	+
JCASim [53]	Java	2001	+	+	+	+	+	+	+	+	+
MCell [54]	Windows	2001	+	+	-	+	+	+	+	-	+
ParCeL-6 [55, 56]	UNIX/Linux or Windows	2004	+	+	+	+	-	+	-	+	+
SIMP/STEP [57, 58]	UNIX/Linux or Windows	2003	+	+	+	+	+	+	+	+	+
Trend [59]	UNIX/Linux or Java	2002	+	-	-	-	+	+	+	-	+

Table 2.1: A comparative survey of existing cellular automata modelling environments. The first column contains the name with references, the second column is used for information about the target platform, the third column gives the year of the latest version. Remaining columns contain grades for the environment’s conformance with universality/customizability, extensibility, modern technologies, optimization, user interface, description language, visualization, analysis, and capability of reproducing respectively. Requirements are listed in the same order as in the last list of Sect. 2.1.

the ability to use completely irregular grids with cells of miscellaneous shapes. The spectrum of the functionality is extremely rich. Transition rules are to be defined using a C-like language [12] with the help of built-in visual programming means. Written rules are translated to C++ [13] source and compiled into an executable code for better performance. Despite the functional richness, all layers seem to be 2D only. There is an example of using of this environments for the dynamic urban simulations [67].

CAM, the “Cellular Automata Machine”, represents a single instruction multiple data (SIMD) [68] hardware implementation of the modelling environment. CAM-6 [17] is a PCI-device which is to be plugged into an iX86 workstation under the PC-DOS operating system government. It supports 256×256 2D grids with Moore, von Neumann and Margolus neighbourhoods [17]. Latest CAM-8 [45] device works in tandem with Sun workstations via CBus and is controlled by the accompanying STEP software (a predecessor of SIMP/STEP project which is also presented in this

survey). CAM-8 supports 3D grids. There is an ability to extend the grid by using multiple device specimens. Visualization is performed by the XCAM utility. Transition rules are to be programmed using a dialect of Forth language [69] supplemented with necessary routines. For each transition function the machine precompiles a full lookup table. However, multiple instruction multiple data (MIMD) architectures are more flexible than SIMD for implementing cellular automata, as they allow to deal with irregularities on a microscopic level of the algorithm asynchronously and to efficiently simulate heterogeneous systems. On a higher level of abstraction it is possible to synchronize the parallel components of a MIMD system explicitly as this is the only way to maintain global invariance of cellular automata.

CAMEL, the “Cellular Automata environMent for systEMs modeLling” was designed to perform computations on a net of transputers or using MPI. It supports grids of up to three dimensions and complex neighbourhoods. The cell’s state can be represented by an instance of a data structure composed of basic types. For the definition of cellular automata it uses a specialized language called CARPET (“Cellular Programming Environment”) [31]. A program written in CARPET consists of a declarative part and statements. The language is clear and successfully hides implementation issues coming from a parallel architecture complexity, allowing to describe automata and rules in general terms. IVT (“Interactive Visualization Tool”) was added to CAMEL in order to improve the data visualization. This tool was used in practice for simulation of lava flow, landslides, highway traffic, for image processing, and genetic algorithms [70]. It was also used for soil contamination modelling [71]. Later, the same group switched to the development of the CAMELot project [51], which is discussed below.

By coincidence the name of the project, created by the author of this thesis (**CAME&L**) is very similar to the previous one. Nevertheless in this case it means “Cellular Automata Modelling Environment & Library”. This project will be discussed in much more details in Sect. 2.3, but here we will review it briefly. The key idea was to create a universal and extensible facility, which supports parallel and distributed computing without any target problem domain specialization. This was achieved by usage of a cellular automata based computational experiment decomposition (see Sect. 2.3.1). The software allows synchronous, asynchronous, probabilistic, heterogeneous, and any other type of cellular automata with arbitrary grids, neighbourhoods or type of cells’ states. Even if a particular functionality or distinct automata type is not included into the basic software package, one may add it and make a new solution immediately available for the community. CAME&L users can be classified into researchers who are just creating solutions from the building blocks they have, and developers who enhance the set of blocks for themselves and everyone. All the research presented in Chapters 3 and 4 was done with CAME&L. This demonstrates its usability for theoretical and practical research on one hand, and for one-, two- and three-dimensional problems on the other hand.

As a descendant of CAMEL, **CAMELot** (“CAMEL Open Technology”) also uses the CARPET language [31] for the experiment description and MPI for the simulation execution. The software represents an environment for programming and seamlessly parallel execution of cellular automata. It has a graphical user interface for experiment

setup, control and visualization. It also includes a customizable tool to produce traces of the simulation in a specified format thus allowing to post-process the output of the experiment by means of external utilities. It also supports profiling capabilities. The simulator is flexible with regard to cellular space size and dimension (from 1D to 3D), possible cell's states, neighbourhood and rules. The program, written using CARPET, is translated and compiled into a UNIX/Linux executable file. The experiment setup preparation consists of editing of a text file. The spectrum of known applications of this environment is close to CAMEL's area. For example, it was used for the landscape dynamics simulation [72].

The **Cellular** software consists of a programming language (Cellang 2.0), associated compiler (`cellc`), virtual machine for a execution (`pe-scsm`) and a viewer (`cellview`). A program, written using Cellang 2.0 consists of a description and a set of statements. The description determines dimensionality of the grid, data-fields, which are contained in each cell, and ranges of acceptable values for each of them. There are two possible statements: an assignment and a condition test. The only possible data type is integer. The viewer is independent of the Cellang 2.0 language and the compiler. At the same time, the input format for the viewer is identical to the output format of Cellang 2.0 programs. The software supports different grids of arbitrary dimensionality, non-trivial neighbourhoods, and several kinds of boundary conditions.

JCASim represents a general-purpose system for simulating cellular automata with Java [16]. It includes a standalone application and an applet for web presentations. The cellular automaton can be specified in Java, CDL [30] or using an interactive dialogue. It supports 1D, 2D (triangular, square, and hexagonal) and 3D grids, different neighbourhoods, boundary conditions (periodic, reflective, or constant), and can display cells using colors, text, or icons. Initially CDL was designed to describe the hardware, which simulates homogeneous structures, but it can also be applied in programming as a powerful and expressive tool. JCASim allows any constructions acceptable in CDL. For example, like in CDL, cell's state can be represented with theoretically unlimited amount of integer and floating-point variables. With the package CAComb the software allows to simulate cellular automata on several machines in parallel. The CAAnalysis package incorporates automatic analysis capabilities as mean-fields calculation, for example. This tool found applications for generalizing surficial geological maps for scale change [73] and it is also used for educational purposes.

MCell or "Mirek's Celebration" is a very small and simple Windows application which supports 2D grids and no parallel or distributed computing. But despite of this its effort is great, because it can easily show the simplicity, beauty and power of a cellular automata to people who are far from this field of science. This is possible due to the successful graphical user interface which is clear for non-specialists, and a wide library of examples. Transition rules can be defined using the interface means or by creation of external dynamic-link library.

ParCeL-6 ("Parallel Cellular Languages") represents a multi-layer cellular computing library for multiprocessor computers, clusters, and Grids. The goal was to decrease the development time for fine-grained applications. It is implemented in

C [12] and can be linked to C and C++ [13] programs. There are two subversions of the software: ParCeL-6.1 for architectures supporting shared memory paradigm and ParCeL-6.2 for architectures supporting the message passing approach. The cluster version of ParCeL-6 was developed in the framework of the Grid-eXplorer project. High level generic parallel neural model of computations allows smart programming for numerous computing units. ParCeL offers the extended cellular programming model and maps “small” computing units on “big” processors of parallel machines. When a cell is created, a host processor is pointed out and a unique registration number is associated with the cell. This number allows to identify it in a cellular network. Finally the cell is created directly on its host processor and executes its computing cycle on it. The software is also able to perform automatic parallelization of the source code for the multiprocessor machines. It was used in several numerical methods libraries. As authors claim, it was designed mainly for artificial intelligence applications [56]. Therewith, ParCel has been used for the cortically-inspired sensorimotor control simulations [74].

SIMP/STEP is a general-purpose software platform, which includes a language for cellular automata, lattice gases, and a “programmable matter” [75] definitions. It is based on Python language [76] and suites for the wide range of problems. The software consists of two parts: SIMP is the user environment built on STEP, the applications programming interface, which separates conceptual components from implementation details, optimization routines etc. The software supports parallel computing technologies, has visualization and analysis capabilities. SIMP supports 2D rendering, but there are some experimental hooks for the 3D rendering, using VTK [66]. The application of this environment for the lattice gas methods is presented in [58].

Trend is 2D cellular automata programming environment with integrated an simulator and a compiler, which produces virtual machine code for the evaluation module (under UNIX/Linux only) or Java machine [16]. It has several unique features like simulation backtracking, conflicts catching, flexible template design and others. The Trend language allows user-defined terms, symmetrically rotatable statements and other constructions specific for the cellular automata programming. The software supports arbitrary neighbourhoods within 11×11 region around the center cell. Cell can be in the state which is coded by unsigned integer variable. The project does not support any parallel computing technologies.

2.3 Cellular Automata Based Problem-Solving Environment Case Study: The CAME_&L Project

CAME_&L [48–50] resulted from a collaboration between the Saint-Petersburg State University of Information Technologies, Mechanics and Optics (Russian Federation) and the Section Computational Science of the University of Amsterdam (The Netherlands).

2.3.1 Cellular Automata Based Computational Experiment Decomposition

The initial idea of “CAME&L” was to distribute the implementation of a computational model among functional parts. Any simulation is to be assembled as a set of interacting components of definite types. A researcher will be able to use components in miscellaneous combinations and tune them to add arbitrary functionality to the experiment.

Consequently cellular automata based computational experiment decomposition [48, 49] was offered. It was decided to distinguish five types of the components.

Grid implements the visualization of cells’ states and the navigation among the cells.

It does not actually store any state data, so this distinguishes it from the “grid”, mentioned in the cellular automaton definition in Sect. 1.3. It was done to strongly distinguish between a data itself and its visual representation.

Datum provides the grid’s state storage, basic operations over the states of distinct cells and some aspects of the visualization. In visualization area it can define an association of cell states with colors and a custom single cell drawing routine.

Metrics defines the relationship of neighbourhood, puts coordinates in correspondence to each cell, and provides distance measurement functions. Implementation of a metrics as a separate component instead of entrusting its functions to a grid or a datum allows to use non-standard coordinate systems, like generalized coordinates [26], explained in Sect. 3.2.

Rules describes the computations and controls an iteration. The terms “rules” and “transition function” are not synonyms. Components of this type define much more: a method of parallelization, methods of optimization (if any), many other aspects of the execution, including the transition function among all of this. Such components also allow

- to handle starting up an experiment (proceed an initialization);
- to determine and check the criteria of termination of an experiment;
- to handle finishing an experiment (proceed a finalization);
- to define special tools for checking, changing, pre- and postprocessing grid’s state;
- to define output parameters for further studying with the help of analyzer components (see bellow).

Analyzer allows to keep an eye on definite output parameters of the experiment, draw graphs, create reports, monitor values, etc.

The union of compatible grid, datum and metrics define a “functionless” cellular automaton. Addition of a component of the fourth type will form a cellular automaton that can perform a computational experiment. Only single instances of components

of first four types are able to participate in a simulation. At the same time it can involve arbitrary amount of analyzers (if any).

Components are continuously interacting during the computational experiment to do the work together. Obviously, any given component can cooperate only with suitable counterparts. So, each one have to pose its properties and requirements, which are interpreted as compatibility conditions. For analyzers such conditions are trivially based on the data type of an output parameter to be examined. For the other four types of components a special language of logical expressions is used to describe properties and requirements. In this case requirements represent conditions imposed on properties.

The last, but not the least is that each component is to have specific user interface, declaring a set of available input parameters, which allow to setup the component for the particular problem or in accordance to user's needs and preferences.

2.3.2 Software Design

All basic statements, listed in Sect. 2.3.1, can be provided with the help of an object-oriented programming paradigm. We used C++ [13] to implement CAME_&L. Windows was chosen as a target operating system. Consequently each component should be represented as a dynamic-link library, developed in the framework of the predefined programming interface. The component's library has to contain the class, which implements the functionality corresponding to one of five types, listed in Sect. 2.3.1.

As a result, the CAME_&L software consists of three conceptually and functionally interconnected parts:

CADLib or "Cellular Automata Development Library" is a C++ class library, which is designed to present an easy-to-use and rich set of instruments for implementing computational models. It provides basic classes for all types of components, parameters (input and output), and other concepts. The class hierarchy of this library is shown in Fig. 2.1 (main classes), and in Fig. 2.2 (auxiliary classes).

Standard components are the most common building blocks for computational experiments, which can be considered as both: ready-made solutions and examples for creation new specimens. They can be reused, inherited*, and extended to fulfil the needs of a researcher.

Environment is the application, which provides user interface for simulations and research with the help of cellular automata. It gains an access to a set of tools for the model control, study and analysis, cluster arrangement, workstations management and many other purposes. It is important to note that the environment itself contains no computational model functionality, but allows to execute components' libraries in a definite software environment.

The tight interconnection between the cellular automata based computational experiment decomposition and the CAME_&L software is illustrated by Fig. 2.3. Three essential parts of the software, mentioned above, are shown in grey.

*Within the scope of the object-oriented programming paradigm.

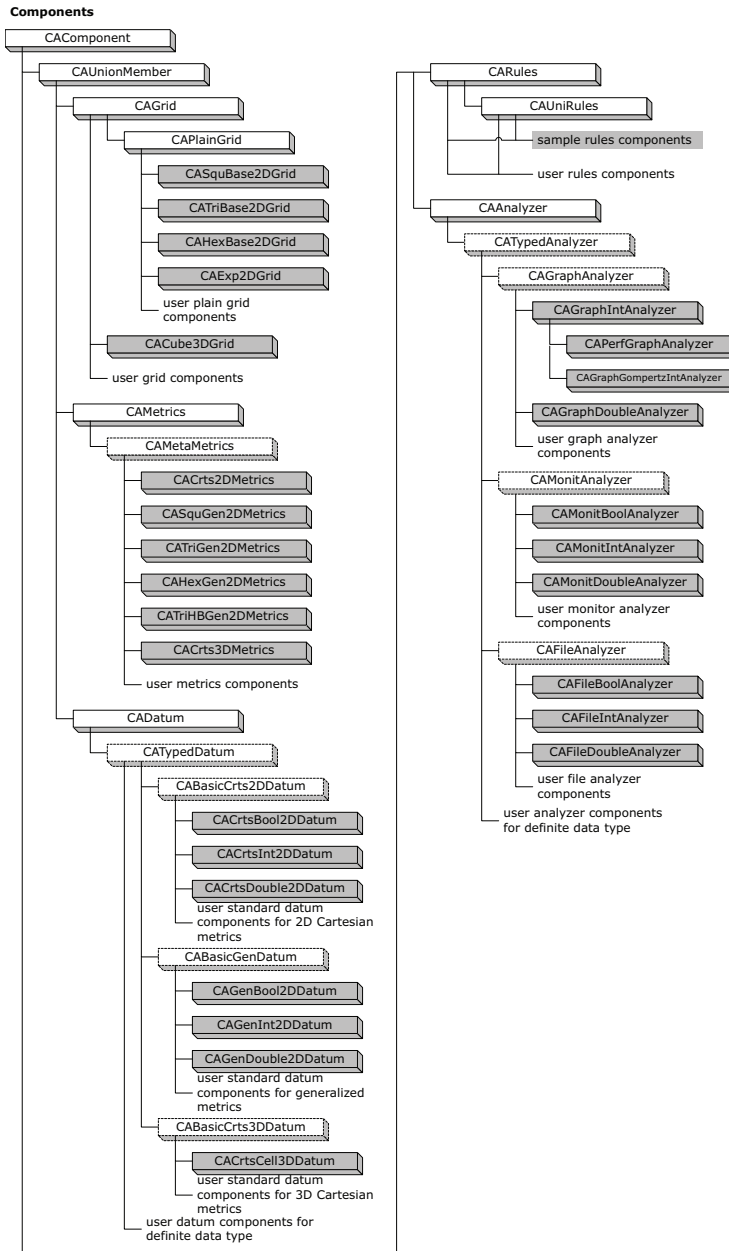
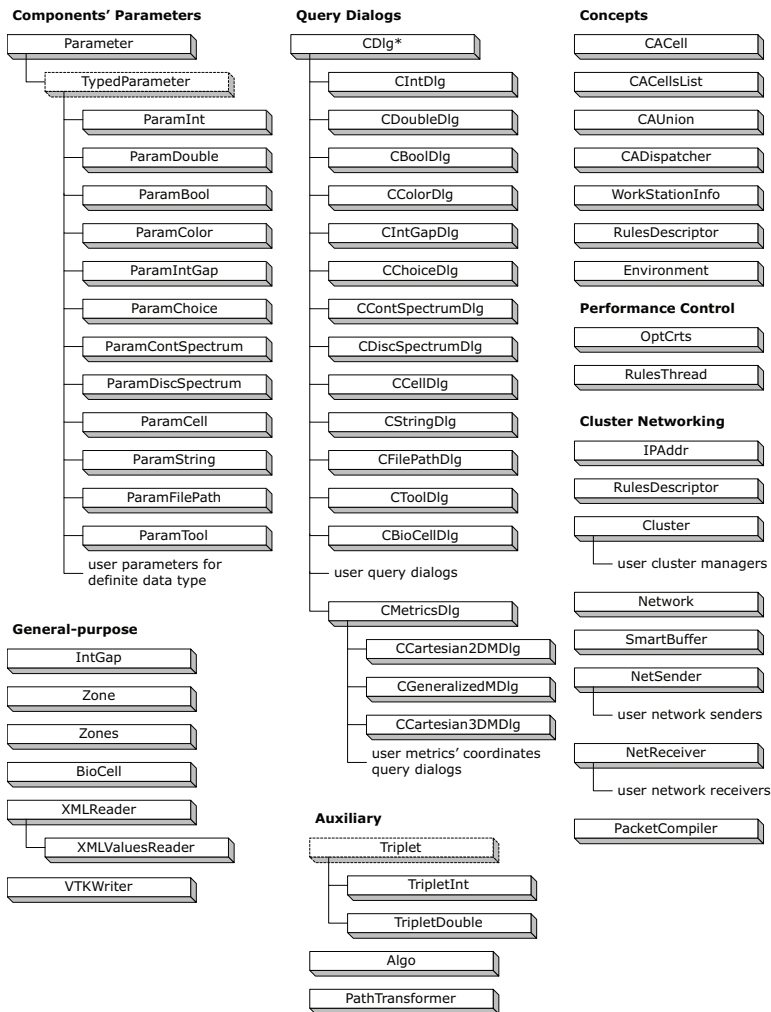


Figure 2.1: CADLib 1.1 β classes hierarchy chart – components classes and their descendants. Each class is shown as a box. Class templates are shown as boxes with dashed boundary. Grey boxes correspond to classes, which are not members of CADLib, but of the standard components set. Appropriate places for further extension of this hierarchy are marked out as a text without boxes.



* - classes, which are inherited from Microsoft Foundation Classes (MFC) library members

Figure 2.2: CADLib 1.1 β classes hierarchy chart – auxiliary classes. Each class is shown as a box. Class templates are shown as boxes with dashed boundary. Appropriate places for further extension of this hierarchy are marked out as a text without boxes.

2.3.3 Features

The most significant features, which distinguish CAME&L from other cellular automata modelling environments are listed and discussed below.

- The main goal was to achieve the universality in the terms of the target problem

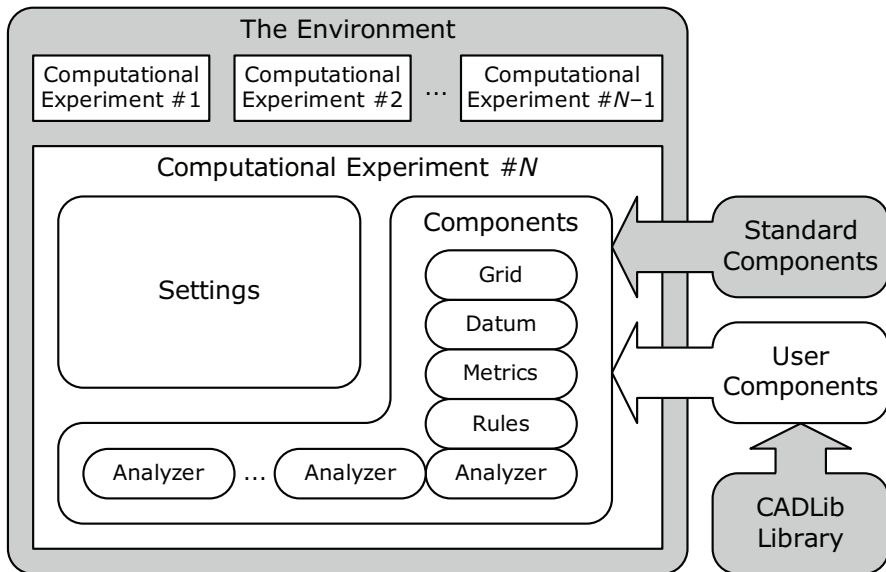


Figure 2.3: Cellular automata based computational experiment decomposition in the framework of CAME_{&L}. Three essential parts of the software are shown in grey.

domain. The software allows to perform experiments in fields of lattice gas, language recognition (only specialized environments allows to use strings or text as cells' states), thermal conductivity, phase unwrapping, cellular biology, social phenomena and many others that allow formulation in terms of cellular automata. There are no intrinsic limitations to the type of cellular automata which can be modelled.

- The environment provides the ability to implement interactions between cellular automata. This was the reason to use a multi-document interface for the software and gives an opportunity to create multiscale computational models.
- The software allows to perform parallel computations on multiprocessor and cluster platforms. Amount of supported architectures can be even enriched thanks to the extensibility and because rules components fully control the execution and parallelization.

The cluster can be created from a subnet of a Windows network. The roles of workstations involved in it can be customized. For the information interchange over the network a specialized Commands Transfer Protocol (CTP) [46, 77] is used.

The trivial implementation of cluster networking is based on the assumption that one workstation involved in the experiment has a “master” status. On each simulation step master broadcasts an order to all other workstations (including himself) to perform an iteration. Workstations immediately interchange the

information about the states of outermost cells (their amount depends on the neighbourhood radius), which can be needed for the transition function calculation on another node. For complex neighbourhoods or other special cases there is a routine to request a state of a single cell from any remote workstation. After this each node starts its subexperiment. At the same time the master is waiting for a signal of step completion from all participants. After this signal has been received, next step can be started. Master node is also participating in the computations, running both master and slave processes.

We performed tests on the example of the Game of Life [78] simulation on the 201×201 grid of squares, parallelizing it over two, four and eight nodes. We compare time, needed to execute 100 steps of the not optimized plain algorithm. The results of simulations and the relative speedup are summarized in Table 2.2.

	<i>1 node</i>	<i>2 nodes</i>	<i>4 nodes</i>	<i>8 nodes</i>
Execution time (in seconds)	22.23	12.42	7.15	4.32
Relative speedup	1	1.71	3.11	5.14

Table 2.2: Execution time for 100 steps of the Game of Life on 1, 2, 4, and 8 nodes.

The default implementation of cluster networking in CAME_&L is straightforward and trivial. It is oriented more to be an example than to be suitable for large-scale tasks. In future we are going to provide a choice between several cluster management routines. At the same time this can be done by users, according to their needs.

- The environment has a rich and clear user interface (see Fig. 2.4). It supports, for example, undo-redo functionality, clipboard operations, printing capabilities, picture capturing functions (to illustrate publications[†]) and many others.
- The environment allows to store experiments descriptions in XML format (files with “camel” extension), which is advantageous for information interchange [79]. States data is stored in separate files (with “cdata” extension), which are usually huge and can be compressed with BZip2 algorithm [80, 81] on the fly. Experiment description files contain only references to the cellular data files, so they can be substituted or removed for opening the documents in the “safe mode”.

The following list shows why and how the CAME_&L project corresponds to the cellular automata modelling environment requirements, discussed in Sect. 2.1.

1. As explained above, **universality** is achieved using the cellular automata based computational experiment decomposition, allowing to compile systems of even wider class than “cellular automata”. The **customizability** is provided by the mechanism of input parameters, which allows to tune each component.

[†]All figures in Chapters 3 and 4, which represent a grid’s state, were produced with the help of this function.

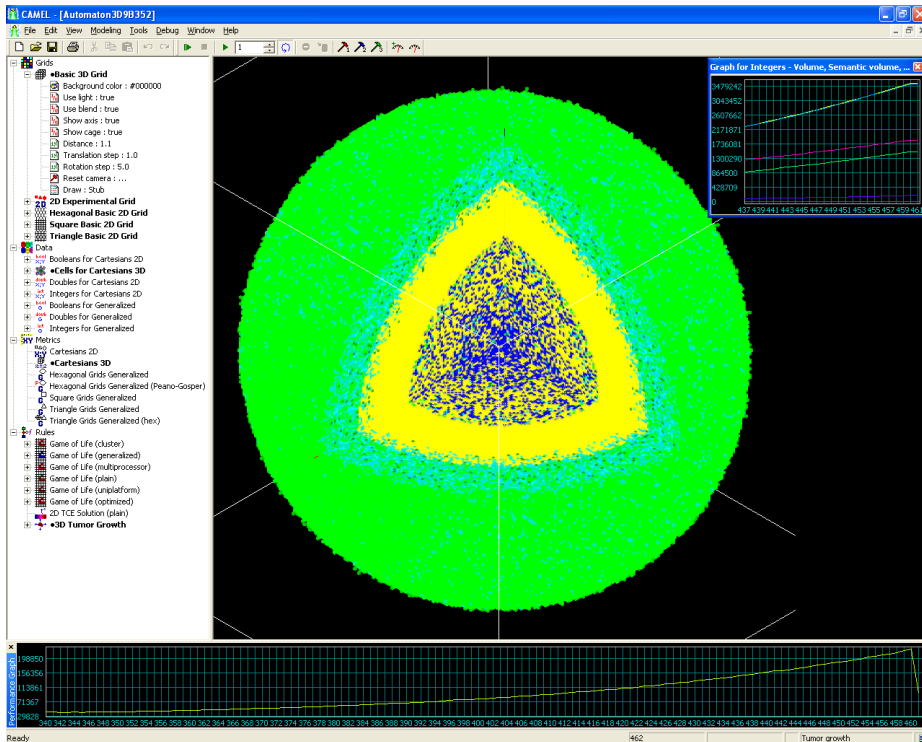


Figure 2.4: The screenshot of CAME_&L environment main window, running tumour growth computational experiment (see Chapter 4), being studied with the help of two analyzers: graph of the amounts of different kinds of cells (at the upper right corner) and the performance graph (at the bottom).

2. The fact that the environment represents just the “firing ground” for components, which implement the whole computational model, provides **extensibility** in any direction.
3. The software supports **modern computational technologies**. Usage of CTP[‡] as a networking vehicle was not a successful idea. CTP is an attempt to implement message-passing using the pure basic UDP [82] mechanism. Due to the fact that UDP is not reliable, reliability was provided on the software level. UDP-packets are very small, this leads us to the necessity to implement large blocks sending mechanism. This all, being implemented on the higher level, brings significant overheads and has bad performance for large blocks interchange. Interchange of small messages is performed on the fastest UDP speed, when large blocks are spreading slower than with TCP [82]. This is illustrated by Fig. 2.5. But the main bottleneck is in the incompatibility with standard

[‡]Now CTP version 1.4 is used in CAME_&L .

mechanisms. At the same time the extensibility of the software allows to extend the range of networking mechanisms with, for example, MPI [63] or Grid protocols [14, 43]. This however is future work.

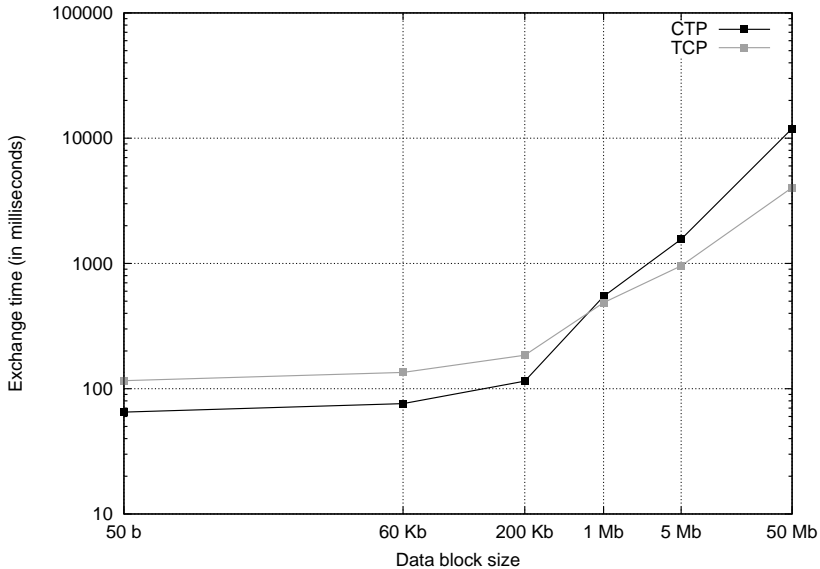


Figure 2.5: Comparative study of CTP (black line) vs TCP (grey line) networking performance while interchanging data blocks of specified sizes.

4. Basic **optimization** capabilities, as the zonal optimization, are included into CADLib. There is a set of routines to create update and look for the parallelepipedic zones, in which dynamic behaviour changes of cell's states may occur. Grid is not considered outside of these zones. At the same time, zones may change their dimensions with the speed of the neighbourhood radius per step. This optimization is trivial and can be used as a starting point for more complex algorithms.
5. Benefits of the **user interface** were discussed above.
6. The **language of the experiment description**, used to determine and store the simulation setup, is a specialized subset of XML [79]. Document files contain names of involved components, values of all parameters and references to cellular data files. They can be easily modified using an arbitrary text editor or the CAME&L environment.

The key issue is the language of a transition function description. By default it is C++ [13]. One could argue that the ability to use C++ is a too complicated skill to demand from a researcher. This is true, but at the same time this is in line with the demand of an extensible universal environment: the scope

of the rules basic class is much wider than just transition function definition. So one can create a rules component, which represents a parser for the specific automata description language. In this case the descriptive code, written in this language, will act as an input parameter for the parser. This means that a single rules component is able to implement not just a single transition function, but a class of such functions, which are expressible on the computations description language. This allows to incorporate arbitrary amounts of specific languages and provide specialists from definite fields of the research with the component, which supports necessary abstractions from a given domain. The code snippets, rich set examples, scripts and the CADLib itself are provided to make such components creation simpler.

7. Due to extensibility, the richness of **visualization** capabilities is high. The basic 3D grid is implemented, using pure OpenGL [83], but one can provide the software with the VTK [66] visualization facility.
8. The software supports and even is oriented to an **analysis** functionality. Due to a very low level of possible user access, there are no limitations on the palette of analysis facilities. Rules components are reporting the set of analyzable output parameters, which are decided to be subjected for the research. A wide range of analyzer components, which allow to study these parameters, can be extended for the needs of the concrete scientific process.
9. **Reproducibility** is provided by the fact that the full simulation setup and the cellular data can be stored in files, which contain no system specific information and are portable. An important note is that XML files are generally handy for a structured data interchange.

2.4 Conclusions

The main goal of problem solving environments has always been to make a researcher more productive and the computations more effective. Appropriate tools may drastically reduce the costs for building new solutions as well as for maintaining existing ones. The development of these facilities is itself a significant engineering task, requiring a considerable investment of time and resources.

At the same time it is very important for an effective use of cellular automata for computational science on parallel machines to develop high-level problem solving environments based on the cellular computations paradigm. These tools may provide powerful solutions for researchers and engineers that need to implement real-life applications. This approach allows scientists to concentrate on "how to model a problem" rather than on architectural details.

Creation of a universal, extensible and handy cellular automata based modelling environments needs a deep study of both fields: typical functionality of problem solving environments and general requirements for cellular automata modelling. The tool, which was made according to the requirements of both fields accounting existing software at once, represents a powerful facility for a research.

The following two chapters will present examples of CAME&L usage to produce scientific results, solving both theoretical and practical problems in one-, two- and three-dimensions.