Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Dottorato di Ricerca in Informatica – XXII Ciclo
Anno Accademico 2008–2009

Dissipative Multilayered Cellular Automata facing Adaptive Lighting

Andrea Bonomi
Ph.D. Thesis

Thesis advisor: Prof.ssa Stefania Bandini
Thesis tutor: Prof.ssa Carla Simone

Contents

Acknowledgements

I would like to gratefully acknowledge my advisor Prof. Stefania Bandini who gave me the opportunity to work on amazing projects. I wish to thank Prof. Carla Simone for her support in the Ph.D and for valuable feedbacks. Moreover, this thesis would not have been possible without the support of the Acconci Studio and Egicon srl.

I am grateful to all my friends from LIntAr, in particular Ettore, Fabio, Giuseppe, Glauco, Ivo, Matteo, Paolo, Sara.

Finally, I am forever indebted to my parents and Gemma for their understanding, endless patience and encouragement when it was most required.

...computers in the future may have only 1000 vacuum tubes and perhaps weigh only 1.5 tons.

Popular mechanics, 1949

1 Introduction

1.1 Motivation

CELLULAR Automata (CA) were introduced by John von Neumann as an environment for studying self-replicating systems (von Neumann, 1966). They have been primarily investigated as a theoretical model and as a method for simulation and modeling of complex systems (Weimar, 1997b). CA are a class of spatially and temporally discrete mathematical systems characterized by local interactions (Wolfram, 1986b). Even if the interaction is based on simple local rules, the resulting structures from the CA evolution may be extremely complex (Wolfram, 1994, 1984a).

CA are also an abstract and formal model of the cellular systems, a wide class of systems present in nature. Such systems are composed of several interacting elements (i.e. cells) acting with a certain degree of independence. The collective behavior (e.g. the respond to the stimuli from the environment) is the result of the local interactions.

The ability to sense and respond to physical stimuli is of crucial importance to all living organisms (Telewski, 2006). This work is inspired by a particular type organisms: the plants. The relative immobility of plants as compared with animals has naturally provoked a dependance upon their ability to sense and respond to subtle environmental signals (Jaffe et al., 2002). However responses to the environmental stimuli are found over the entire range of the plant kingdom.

A beautiful example of such global behavior obtained by simple interactions is the *Heliotropism*, a phenomenon in which the leaves or leaflets adjust their position with respect to the direction of incoming solar radiation (Ehleringer and Forseth, 1980). As a result of these movements, leafs an sun rays become perpendicular (maximizing the amount of absorbed radiation) or parallel (reducing the transpiration). The mechanism of leaf movement involves cells turgor changes. An increase in turgor on one side is accompanied by a decrease in turgor in the opposite side, leading to leaf movement, as shown in Figure 1.1.

The rationale of this work is to exploit this kind of self-organizing behavior present in biological systems to model, design and realize Distributed Control Systems.

Distributed Control Systems, schematized in Figure 1.2, are composed of *nodes*, managing internal resources and interacting with the physical environment (through

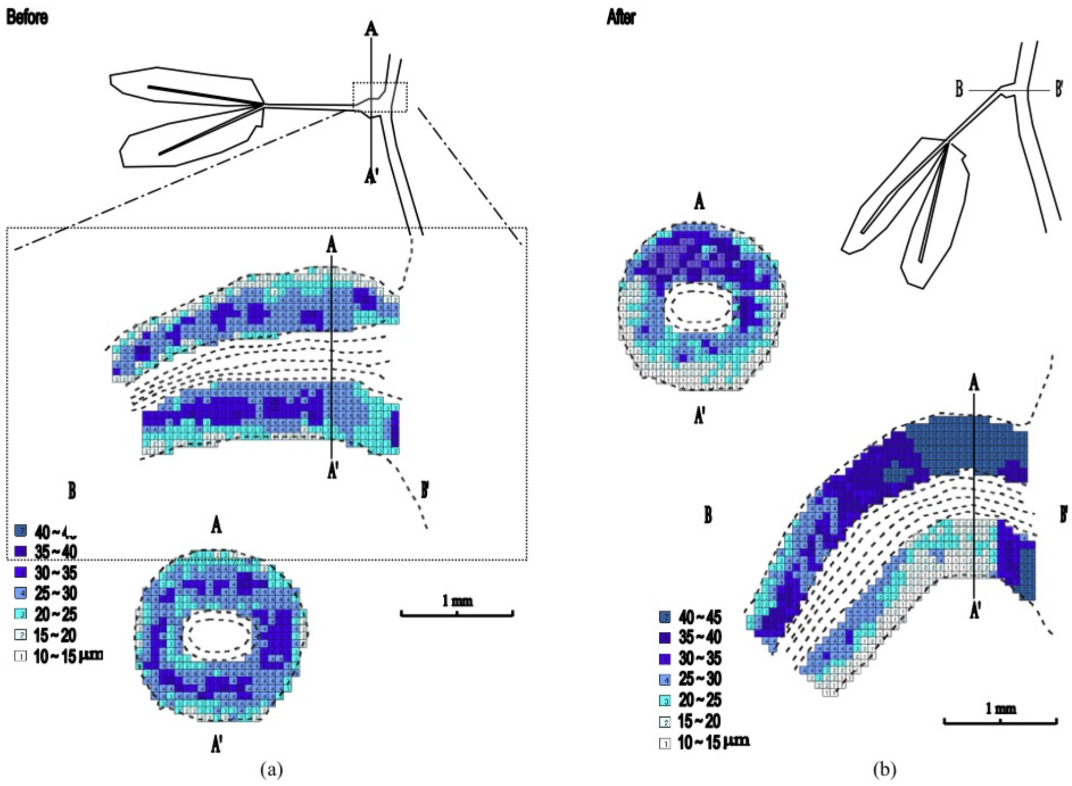


Figure 1.1: *Size distribution of motor cells in main pulvis of Mimosa Pudica (a) before and (b) after the petiole is stimulated by touch. (Source: Taya, 2003, fig. 7, p. 59)*

sensors and actuators) and neighboring nodes so as to obtain the desired overall system behavior as a result of local actions and interactions among components.

For many applications, especially when there are several components characterized by a certain degree of local control and interactions with other elements of the system, a distributed control system approach is suitable and more natural than a centralized control system.

Adaptive Lighting is one of the paradigmatic examples of applications that cannot be always controlled in a comfortable way with a centralized control system. Adaptive Lighting is a broad application field regarding the design and development of lighting systems capable of controlling the light emission according to internal programs and external stimuli.

Interactive installations, stage lighting, and domotics are the major fields of application of Adaptive Lighting. Figure 1.3 shows a few examples of different Adaptive Lighting applications.

Adaptive Lighting is a particular case of Ambient Intelligence (Shadbolt, 2003), that promises, seamlessly integrating computing with the physical world, to give so-

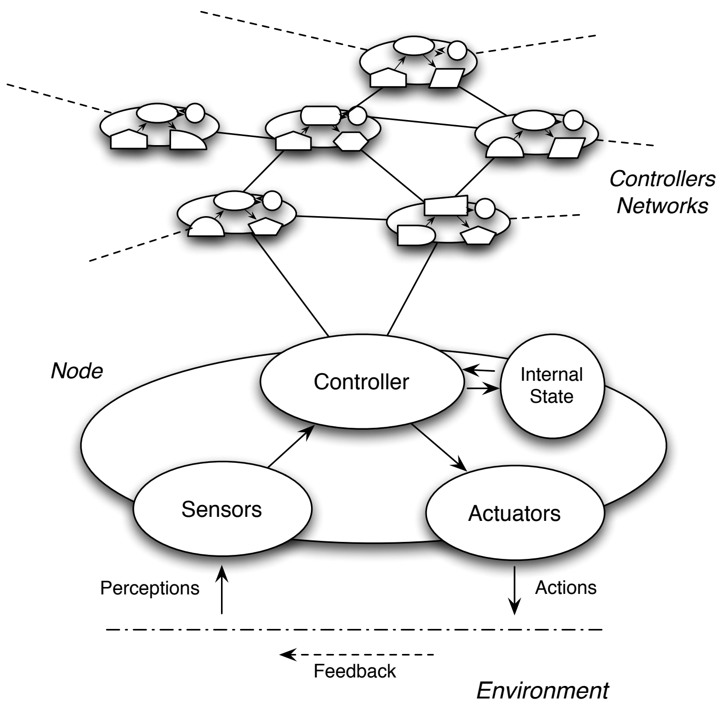


Figure 1.2: *Schematization of a distributed control system.*

ciety an improved living standard, greater security, and unparalleled convenience and efficiency.

In the Ambient Intelligence vision, data and services will be available anyplace, any time to all people. Many services and facilities (e.g. public transportation, traffic control, electricity distribution networks, heating, ventilation, and air conditioning) will become more efficient, integrated and capable. Such systems will be able to interact together and with the people, improving our environment according to our needs.

The main aim of research on Ambient Intelligence is the definition of models and tools for the realization of environments endowed with a large number of electronic devices, interconnected by means of wireless communication facilities, able to perceive and react to the presence of people. These facilities can have different goals, ranging from explicitly providing electronic services to humans accessing the environment by means of computational devices (e.g. personal computers or PDAs), to simply providing some form of ambient adaptation to the users' presence (or voice, or gestures), without requiring an explicit interaction though a traditional computational device.

An Ambient Intelligence system can be viewed in terms of autonomous entities, managing internal resources and interacting with surrounding ones in order to obtain the desired overall system behavior as a result of local actions and interactions among system components. Approaches that take this perspective share a growing interest

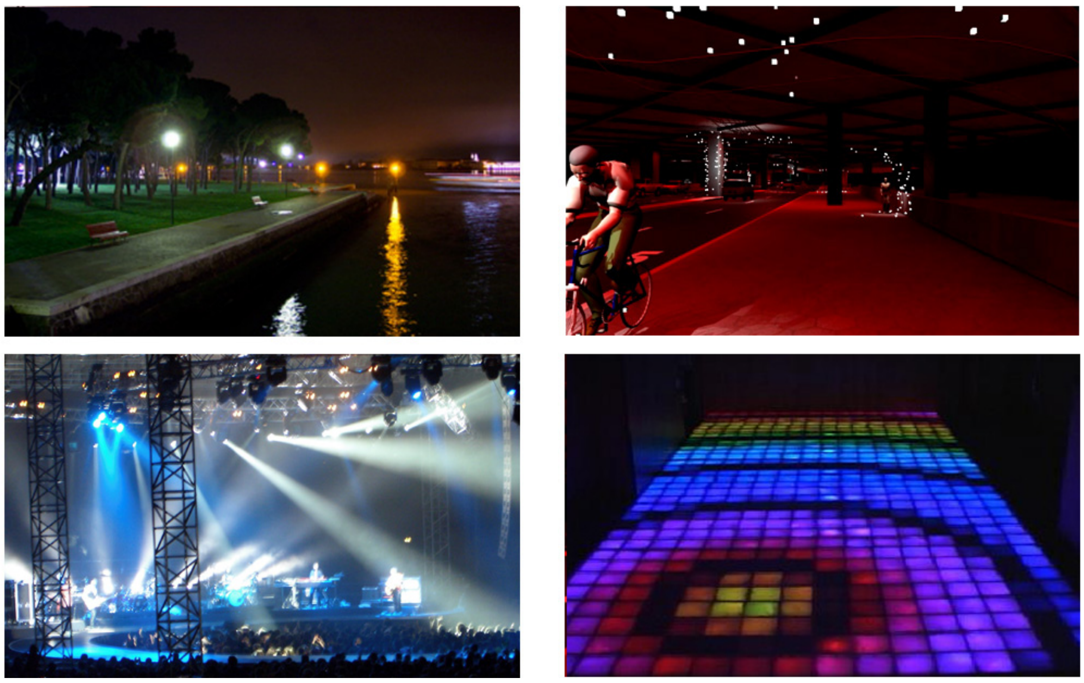


Figure 1.3: *Examples of different Adaptive Lighting applications: public lighting, interactive installation, stage lighting and decorative indoor lighting.*

on models and mechanisms supporting forms of self-organization and management of the components (both hardware and software) of such systems.

Several authors consider that the recent confluence of embedded and real-time systems with wireless, sensor, and networking technologies is creating a nascent infrastructure for a technical, economic, and social revolution (Stankovic et al., 2005). On the other hand such pervasive technology application opportunities, presents itself as problematic (e.g. regarding the privacy, see Kafeza and Kafeza, 2009) and it is not without criticisms (e.g. see Araya, 1995; Friedewald, 2005).

The Ambient Intelligence vision is not the only one to propose the adoption of embedded computational devices to enhance human environments. A similar idea was expressed by Nicholas Negroponte in his book Soft Architecture Machines (Negroponte, 1975). He coined the term *Responsive Architecture* when he proposed that architecture would benefit from the integration of computational devices into built spaces and structures, and that better performing, more rational buildings would be the result. Research in this area has had to do with the ability to adapt the building structure to the needs of the people.

In recent years, a considerable amount of effort has been spent on *intelligent homes*, the emphasis here has been mainly on the development of computerized systems to adapt the interior of the building to the needs of users. This could improve the standard of living for many people, especially elderly and disabled persons. In fact, technology to help the elderly and disabled living independently is one of the major reasons that such technologies began to be developed. Other major applications are integrated security, safety systems, ease of living and comfort, and energy saving.

The starting point of this work is the following question: is it possible to control in a comfortable way Adaptive Lighting systems with a cellular distributed control system in order to take advantage of the cellular systems' qualities (e.g. auto-organization, robustness, self-repair and adaptation)?

Adaptive Lighting is in fact a challenging Ambient Intelligent application and a positive result in the above direction represent a starting point for a more general application even in other scenarios.

1.2 Objectives

In order to address the previously introduced general question, several specific objectives has been defined:

- To define a cellular automata model suitable for distributed control and in particular for the control of Adaptive Lighting installations. This objective implies to understand the Adaptive Lighting domain-specific issues.

- To investigate the effect of the asynchronous nature of cellular distributed control systems, in order to understand the problematics deriving from the adoption of an asynchronous model that is much more adequate to distributed systems than a synchronous one.

- To build a prototype of a real Adaptive Lighting installation in order to verify it the proposed model is suitable. This task requires to deeply understand the light designers' requirements and to create tools helping the designer in defining the dynamic behavior of the system.

- To define a programming language suitable for creating control systems with the proposed model. In order to enable the designer to define by theirself the cells' behavior, a visual programming language is required (visual programming languages are popular in communities of designer and artists).

- To create a cellular execution environment suitable for the class of microcontrollers typically used in distributed control systems.

- To design an Adaptive Lighting system module based on the proposed model suitable for different adaptive lighting installations. Such module is intended as an off-the-shelf solution requiring a set of services enabling the end user to design and construct their own installations.

1.3 Outline of the thesis

Chapter 2 presents the Adaptive Lighting scenario. We present the major fields of application of Adaptive Lighting, the approaches to control such lighting systems and the relevant approaches and technologies.

Chapter 3 gives an overview of the state of the art of Cellular Automata, that were introduced by John von Neumann as an environment for studying self-replicating systems (von Neumann, 1966). CA in fact are the fundamental model for cellular system that will be employed in this work.

Chapter 4 discusses the issues deriving from the asynchronicity in Cellular Automata. Cellular Automata have traditionally treated time as discrete and state updates as occurring synchronously and in parallel. However, several authors (e.g. Paolo, 2000; Thomas and Organization., 1979) have argued that asynchronous models are viable alternatives to synchronous ones and suggest that asynchronous models should be preferred where there is no evidence of a global clock in the model of the system.

Chapter 5 proposes a Cellular Automata model for Adaptive Light: Multilayered Dissipative Cellular Automata (MDCA). The main characteristics of the models are: Asynchrony, Heterogeneity, Multilayered and Openess. Such features are useful for designing systems composed of several distributed interacting components. MDCA can be used both to simulate the behavior of such distributed systems and to control the real installations.

Chapter 6 introduces the Indianapolis Project, an Adaptive Lighting installation of the Acconci Studio, and then it presents the proposed cellular-automata based model.

Chapter 7 narrates the experience of the application of the model. In this chapter we propose a Modular Adaptive Lighting System composed of several independent modules, equipped with proximity sensors, and RGB leds guided by the proposed model.

Chapter 8 contains the conclusion with indication of future work.

Sometimes the best lighting of all is a power failure.

Doug Coupland

2 Adapting Lighting

2.1 Application Scenarios

ADAPTIVE Lighting is a broad application field regarding the design and development of lighting systems capable of controlling the light emission according to internal programs and external stimuli. Adaptive Lighting involves several different aspects from lighting design to hardware (e.g. automated lights, other actuators, sensors). Lighting control is a key component of Adaptive Lighting.

There are several human activities requiring a lighting control based on timing or events. Examples of Adaptive Lighting's application are stage lighting, interactive installation and domotics. In this section we present these three different application scenarios. This section does not represent an exhaustive description of all the aspects of this area but will provide the reader with a basic knowledge of some scenario motivating the presented work.

2.1.1 Interactive Installation

Installation is arguably the most original, vigorous, and fertile form of art today (De Oliveira et al., 1994). Installation Art is a kind of modern art in which artists use the specific setting (e.g walls, floor, lights) as part of composition. Typically the viewers are able to move around the works and interact with the them, so that they become part of that work in that specific moment in time. Installations are usually not intended to be permanent.

Interactive installation is a kind of installation art in which the viewers are able to interact with the artist work. Different kinds of interactivity can be defined. In Hannington and Reed (2002), three types of interaction are presented:

Passive where the content has a linear presentation and users interact by only starting and stopping the presentation;

Interactive when users are allowed to choose a personal path through the content;

Adaptive is the interaction in which users are able to "enter their own content and control how it is used".

Usually, an interactive installation involves the audience acting on it or the installation responding to the people activity. Edmonds et al. (2004) proposed four categories of relationship between the artwork, artist, viewer and environment:

Static where the installation lack of interaction;

Dynamic-Passive when the artwork response is triggered by environmental factors (e.g. sound, light, temperature);

Dynamic-Interactive where, in addition to the environmental factor, the human presence and actions influenced the artwork;

Dynamic-Interactive (varying) where, in addition to the human interaction, the original specification of the art object is modified by an agent (either human or software).

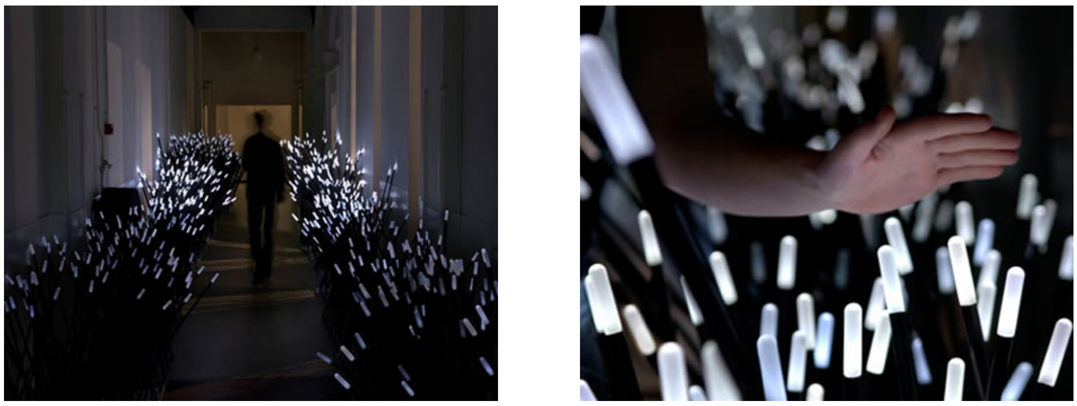
With the technology improvement over the years, artists are now able to create installations involving sensors (e.g. motion sensors, touch sensors, light sensors) and actuators (e.g. lights, monitors, speakers). Two examples of interactive installations are shown in Figure 2.1.

The interactive installations' behavior become more complex, requiring the integration of information from different sources and the coordination of several actuator. Today there are several programming environment created for develop the control system of such installations. The aim of that software is to bridge the gap between the requirements of the artists and the complexity of the development of a control system.

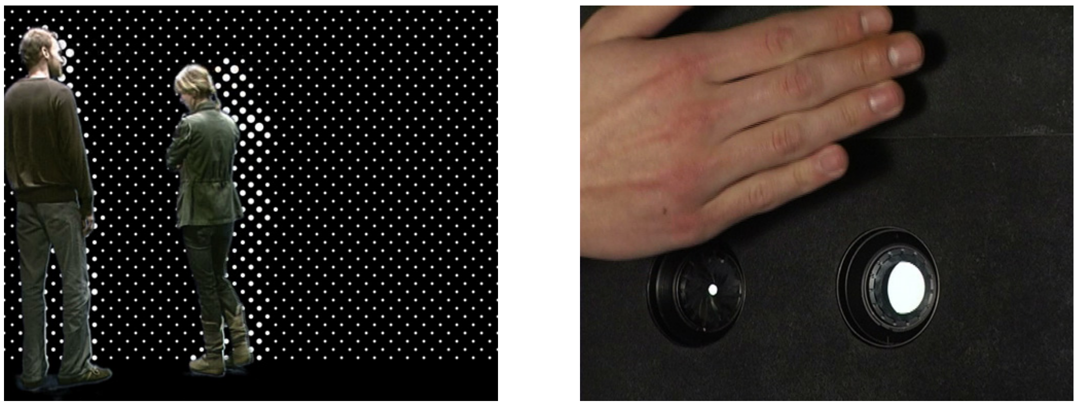
Moreover, complex installations commonly requires specialists with different areas of competence to collaborate with the artist (e.g. architects, electrical engineers, software engineers). The artist could express itself with its own language (e.g. draws, animation, narrative description) and the engineer have to translate such desiderata into hardware and program requirements.

Requirements definition is one of the most difficult tasks for the software developers in the interactive installations. According to Machin (2002), greatest challenges in even identifying what the artist requires. In Marchese (2006) is asserted that the most important part of the process because without a precise understanding of the system requirements it is possible to build a well functioning system that does not perform the tasks requested by the user.

Several authors (e.g. Machin, 2002; Edmonds et al., 2004) suggest that some of the difficulties in requirements elicitation might be due to practical communication problems, since artists and technologists use their domain specific language, terms and concepts and might misunderstand each other or underestimate the importance of issues from the other domain. It is important that both artists and technologists are aware of these properties of the requirements. Requirements might be difficult to capture, vague at the beginning and frequently changeable. Having this in mind will allow choosing the most appropriate software development methods and designing the most suitable architecture of the product (Trifonova et al., 2008).



Dune 4.0 by Daan Roosegaarde is an interactive landscape which physically changes its appearance in accordance to human presence.



Aperture by Frédéric Eyl and Gunnar Green is an interactive facade installation consisting of an iris diaphragm matrix.

Figure 2.1: *Examples of interactive installations.*

2.1.2 Stage Lighting

Stage lighting is a branch of the lighting design concerning the illumination of "live events", such as theatrical productions, dance, opera, concerts, sports events, fashion shows, conferences, TV shows.

Stage lighting is often not only used on stages, but also for permanent installations. Both human environment (e.g. monuments, fountains, squares, bridges, swimming pools, retail stores) and natural environment (e.g. caves, fall) are often enhanced through the use of lighting.

The function of lighting is not only illumination, but also help to set a scene, or focus and direct the audience attention. Examples of different light effects during a concert are shown in Figure 2.2. Light without a precise control is not much use for most situations, and because of this, lights control has long been the critical part of stage lighting.

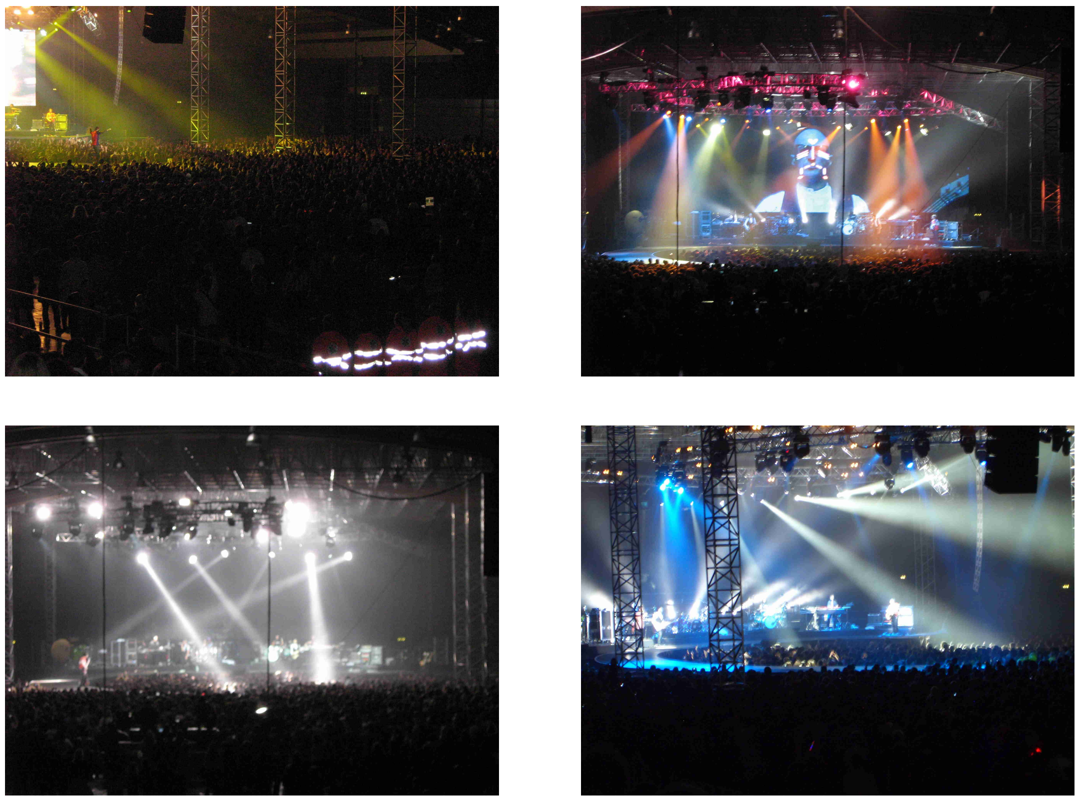


Figure 2.2: *Examples of different light effects during a concert.*

In a typical show, several light have to change their parameter at a precise time or in response to an external event. There are several configurable parameters for the modern lighting fixtures, such intensity, color, and other parameters. Moving lights have many more control parameters than fixed lights: multiple axis positions, gobo[1], shutter, and focus.

[1]A gobo controls the light by blocking, coloring, or diffusing some portion of the beam before it reaches the lens, in order to project the desired pattern onto whatever surface it is pointed at such as a wall or floor.

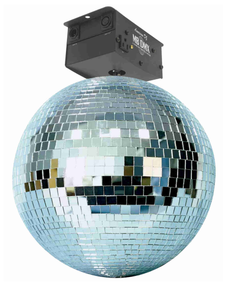
Par Can (Par56 DMX LED RGB)

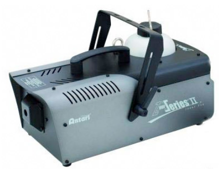
Scanner (DMX–600 Intimidator 1.0)

Moving Light (Chauvet MiNWASH)

Led bar (EUROLITE LED bar 324/10 RGB)

Profile Spot (PS-A014B)


Mirror Ball (DMX Mirror Ball American DJ)


Fog Machine (ANTARI Z-1000II)

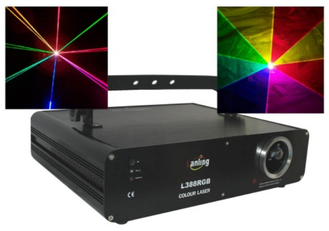

Laser Show (L388RGB)

Figure 2.3: *Examples of different kind of stage fixtures.*

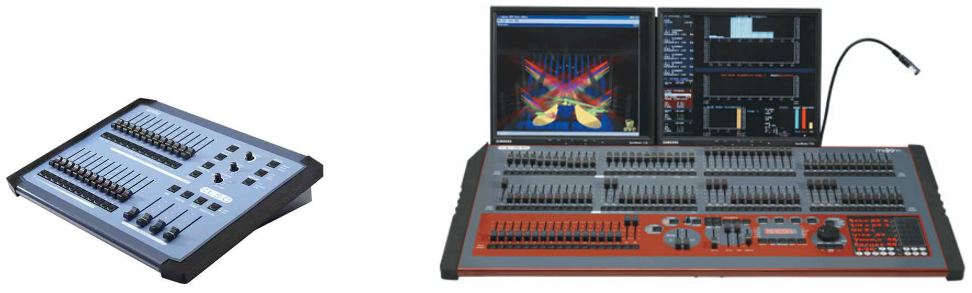


Figure 2.4: *Examples of two different lighting consoles produced by LSC Lighting Systems Pty Ltd. On the left, an entry level console, on the right an high end console.*

Other equipment are generally controlled by the stage lighting system, such mirror ball, laser light show, video server and artificial fog. In Figure 2.3 are shown several kind of stage fixtures.

Laser systems provide spectacular effects for a wide variety of production. The basic concept behind a laser light show is simple: using one (or more) colored, narrow beams of light, a graphical picture is drawn.

Video Servers are the primary source for modern video playback. A video server typically plays back a digital video file from a hard disk. External control is critical for video servers, since they are integrated with other performance elements. Many video servers are capable of a wide variety of control over parameters such as image geometry and playback speed. The video signal, provided by the video server, is delivered to video monitor, projectors or large-format display devices such as LED displays. Professional video monitors and video projectors are generally capable of external control, if only for simple things such as power and lamp on/off and input selection.

There are three parts in a modern lighting control system: a *control console*, the *control data distribution system*, and the controlled devices.

Early control consoles a row of sliders would be set as a “scene” or “preset” and the operator would manually “cross-fade” between the presets. In this preset operation, every parameters have to be entered into the system to be associated to each cue. In more modern consoles, only the changes to each scene need be entered. These consoles are known as “tracking” consoles. Today, for the complex shows are often used fully computerized consoles. In Figure 2.4 are show two examples of lighting consoles.

2.1.3 Domotics

Domotics is the application software and hardware in housing applied to the areas of safety and security (e.g. alarms, surveillance), comfort and self-care (e.g. lighting control, ventilation control, heating control), communication, property control and management (e.g. energy saving) (Allen et al., 2001).

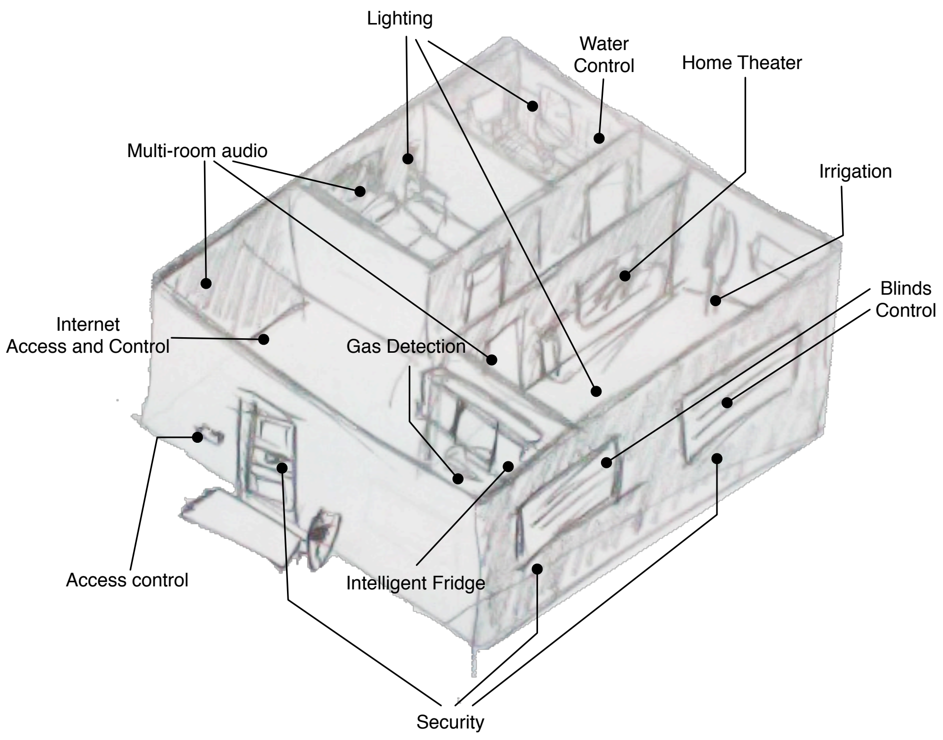


Figure 2.5: *Examples of the domotic house's services.*

Domotics is the field where housing meets technology in its various forms (informatics, but also robotics, mechanics, ergonomics, and communication) to provide better homes from the point of view of safety and comfort (Aiello and Dustdar, 2008).

Domotics evolves from the tradition home automation solutions, that are simple control systems, including set of sensors, switches and actuators, connected together using direct wiring. At the beginning, home automation solutions were provided by one vendor, using one proprietary standard for communication. Successively, several standards have been developed by different vendors. Domotics devices are heterogeneous in all aspects: they are produced by different vendors, have different hardware features, network interfaces, and operating standards. Today one of the biggest challenge for the diffusion of the domotics is the interoperability.

The key concept of domotics is integration of home control, entertainment and computers into one environment. In order to do so, it is necessary to establish communication links between all devices and have a shared protocols understood by all members of the network.

There are several alternative technologies to establish communication links between the devices. The mainly adopted are powerline for networking, dedicated wired networks, wireless networks, and usage of existing wired networks (e.g. Ethernet).

According to Aiello and Dustdar (2008), the key properties to judge a domotic technology and related standards are:

Openness the publicity of the protocol and the possibility of implementing it on any home appliance;

Scalability the possibility of adding and removing devices to and from a home network without affecting its functionalities and its performances;

Heterogeneity the support for different kind of hardware, networks, operating systems, and programming languages;

Topology the way in which devices are connected to one another.

2.2 Design and Implementation

There are mainly two approaches to the design of dynamic behavior for adaptive lighting and interactive installation in general: define the behavior using an ad hoc application to configure a predefined model to achieve the desired effect (e.g. stage lighting control software) or actually writing a program to realize the behavior using a (either general purpose or specialized) programming language. In the following sections we examine both the options.

2.2.1 Programming Approaches

In this section, we introduced the most popular programming languages used for creating multimedia interactive installation. In out knowledge, there are no specific programming language for the lighting control. Moreover these programming languages are suitable for adaptive lighting. Many of them are visual programming languages. Visual Programming refers to any system that allows the user to specify a program in a two or more dimensional fashion (Myers, 1990). Visual programming languages have a long history during which there have been many different languages developed with the common goal of ameliorating the difficulties of programming (Edmonds et al., 2005). There is a common belief that visual languages are easier to use for end users. However, there is no scientific evidence that visual is generally better or easier than text (Goodell et al., 1999).

2.2.1.1 Max

Max[1] is a graphical programming environment for music and multimedia. During its 20 year history, it has been primarily used by performers, composers, artists, scientists, teachers, and students, for creating interactive software and installations. Max

[1]http://www.cycling74.com/

is widely regarded as the *lingua franca* for developing interactive music performance software.

Max is named after Max Vernon Mathews, a pioneer in the world of computer music.

Max was originally developed by Miller Puckette in the mid-1980s at IRCAM (Institut de Recherche et Coordination Acoustique/Musique) to give composers access to an authoring system for interactive computer music. It was first used in a piano and computer piece called Pluton written by Philippe Manoury in 1988. The first commercial version of the program was sold in 1990 by the Opcode Systems and since 1999 by Cycling '74 company.

Max is highly modular, allowing third-party development of new routines. As a result, Max has a large community of programmers who enhance the software with commercial and non-commercial extensions. Extensions to the program can be written as Max patchers, or written in C, C++, Java, or JavaScript.

Most notably, a set of audio extensions, called MSP, allowed for the manipulation of digital audio signals in real-time, allowing users to create their own synthesizers and effects processors.

Max is a data-flow language: Max programs (called *patches*) are made by arranging and connecting building-blocks of *objects* within a *patcher*. These objects act as self-contained programs, each of which may receive input through one or more visual input line and generate output through visual output lines. Objects pass messages from their output line to the input line of the connected objects. Messages can be atomic data types (e.g. int, float, symbol) and more complex data structures (e.g. array, hash table, XML, audio, video)

There are graphical object, including sliders, number boxes, dials, table editors, pull-down menus, buttons, and other objects allowing controlling the program interactively. An interactive installation could be controlled by the viewers through this kind of graphical interfaces. Another possibility is react to external MIDI (Musical Instrument Digital Interface) events, allowing the Max-controlled installation to interface to a large number of MIDI complaint devices. A developer can also create ad-hoc extension to interact to specific sensor (e.g. motion sensor).

It was observed that the diagrammatic layout of the Max is effective because it represents the underlying computational or logical process.The user derives a sense of engaging directly with this process (Edmonds et al., 2005).

2.2.1.2 Pure Data

Pure Data[1] is a free real-time graphical programming environment for audio, video, and graphical processing. Pure Data provides the main features of Max, but is also intended to support the definition and editing of compound data structures in a more

[1]http://puredata.info/

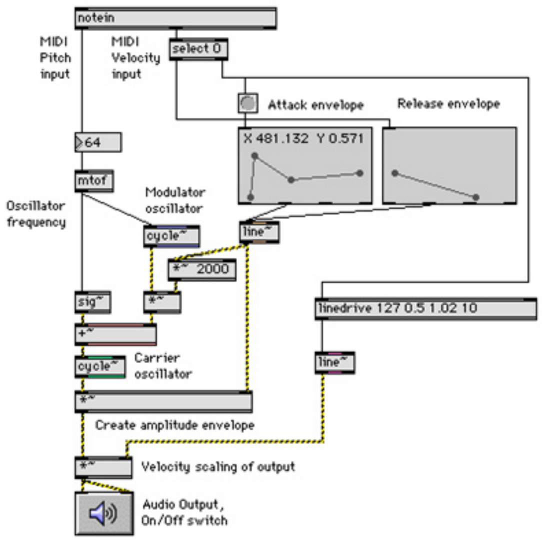


Figure 2.6: *A MAX patch. The program structure and the user interface are presented simultaneously.*

sophisticated way than Max does (Puckette, 1996).

The core of the language is written and maintained by Miller Puckette (the original author of Max) and includes the work of many developers. The work of many developers is already available as part of the standard packages and the developer community is growing rapidly. Recent developments include a system of abstractions for building performance environments and a library of objects for generating and processing video in realtime.

According to Miller Puckette, the most significant weakness of Max is the difficulty of maintaining compound data structures of the type that might arise when analyzing and resynthesizing sounds or when recording and modifying sequences of events of many different types. Also it has proved hard to integrate non-audio signal, video or sensors information for instance.

Pure Data's working prototype attempts to simplify the data structures in Max to make them more readily combined into user data structure.

As shown in Figure 2.7, it is composed of two main parts. The Pd, shown on the left, does realtime computation using a Max-like interpreter and schedule. All the programs (i.e. *patches* and *objects*), including the editor, reside in the address space of Pd. The other process, Pd-gui, talks to the window system through the *Tk* (Osterhout, 1994) cross-platform widget toolkit.

In order to better present the Pure Data program language, we describe the example program presented in Figure 2.8. This program is a synthesizer, generating sounds at different frequencies.

The *metro* object is turned on and off by sending either a 1 or a 0 to its left input line.

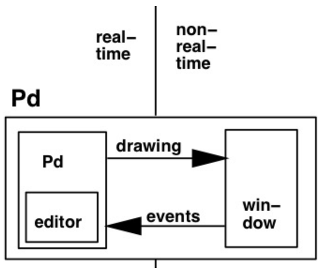


Figure 2.7: *Pure Data architecture, showing realtime and not-realtime modules. (Source: Puckette, 1997, fig. 1, p. 270)*

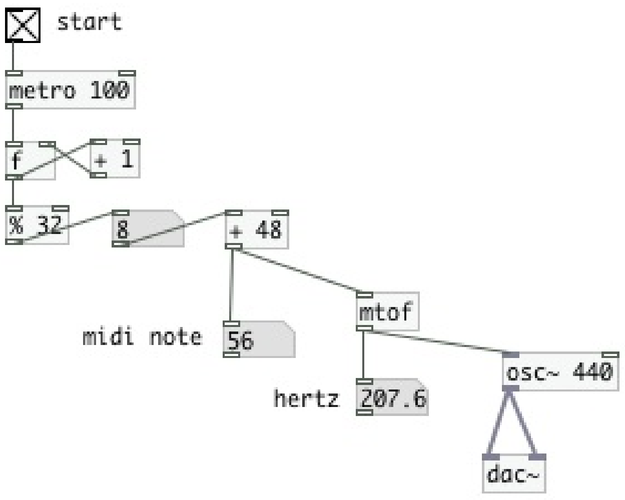


Figure 2.8: *An example of a Pure Data patch generating sounds at different frequencies.*

We use the *toggle* button near the start label to send these messages. *metro 100* is used to send the message *bang* every so 100 milliseconds. *f* is a float variable. Its output is connected to the object *+ 1*, which returns the value of the variable incremented by one. Since the output of the *+ 1* object is connected to the input line of *f*, the value of the variable replaced by the new value. These operations are performed every time a *bang* message is received from the *metro* object, so the value of the variable is incremented by one every 100 milliseconds. The value of the variable is taken as input for the *% 32* object, that returns the value of the variable modulo 32. The output is send both to a gray box near the midi note label and to the *mtof* object. *mtof* is the object which turns a MIDI note into a frequency in Hertz. The resulting frequency (i.e. the output of *mtof*) is sent both to the gray box near the hertz label and to a sine wave oscillator *osc* object, which sends audio to the *dac* (Digital to Analog Converter). The Digital to Analog Converter is the connection to the soundcard. The two connections between the *osc* and the *dac* represent the audio left and the right channels.

2.2.1.3 vvvv

vvvv[1] is a visual programming environment for real-time graphics, video processing and installation control. It was created by the Meso group in Frankfurt, who originally designed it as a tool to design their installation projects. vvvv is free for non-commercial use. Any commercial use requires a license.

vvv is similar in operation to Max/MSP, as shown in Figure 2.9, but focused on visuals and show control. It has functions for controlling a variety of different types of third party devices, including DVD players, touch-screen monitors, gaming devices, switches, position and orientation sensors, MIDI equipment, DMX interfaces, serial port devices, keyboards and (multiple simultaneous) mice.

vvvv file format is XML conform which, allows reading data from a running script as well as setting a script state from itself. In other words a script can manipulate itself. Another feature of vvv is *boygrouping*, that it is a way of setting up a piece to render on a cluster of separate PCs using a master-slave distribution setup, allowing easy multiple-projection output. An integrated Web Server allows direct serving of web content and can be useful for remote administration of vvvv installations.

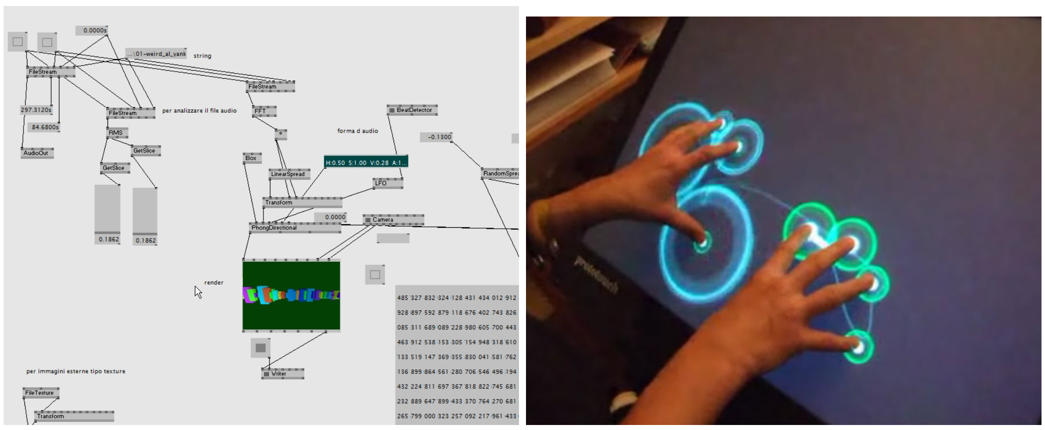


Figure 2.9: *On the left, a screenshot of the vvvv visual programming environment, on the right an example of a multitouch prototype controlled with vvvv created by Chris Engler.*

2.2.1.4 Quartz Composer

Quartz Composer is a real-time visual programming environment developed by Apple Computer. Quartz Composer uses OpenGL, JavaScript, and other technologies to build a developer tool around a simple visual programming paradigm. It has many similarities to Max, Pure Data and vvvv, although its primary usage is for graphical rather than audio processing. It is able to access to many functions offered by the Mac OS X operating system, such as MIDI, Networking (Web, RSS), Audio Input/Analysis,

[1]http://vvvv.org

video and audio filters. The ability to construct interactive video compositions that react to audio or MIDI signals is one of the features allowing the creation of interactive installations with Quartz Composer. A tool called Quartz Composer Visualizer allows compositions to be rendered across multiple screens on a single machine, or even spanned across several machines and displays.

Most of programming in Quartz Composer is done by drawing connections between nodes (i.e. *patches*), twirling dials and entering values into input boxes. As Max, Pure Data and vvvv, each change to the program is immediately reflected in the viewer, without recompiling. As show in Figure 2.10 the interface of Quartz Composer is simple and intuitive. Each *patch* is similar to a subroutine in a traditional programming language. The *patch* can receives input from other *patch* and produce some results. Circles on the left side of a *patch* represent the accepted inputs, circles on the right side are the *patch* outputs. For example, the Random patch will accept *Min* and *Max* parameters and use them to create a *Value* output,

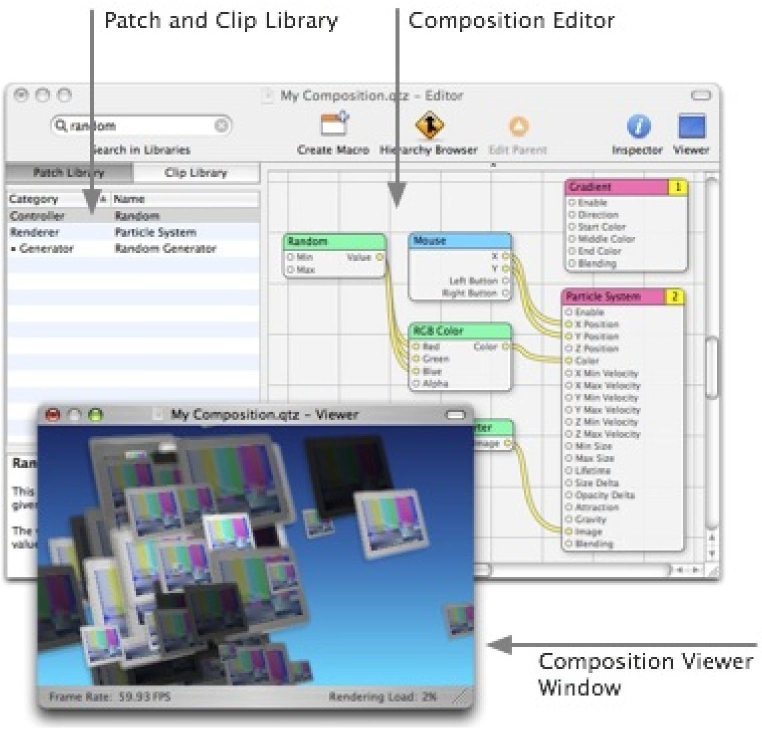


Figure 2.10: *A screenshot of the Quartz Composer User Interface.*

2.2.1.5 Processing

Processing is an open source programming language and environment built for the electronic arts and visual design communities. The project was initiated by Ben Fry and Casey Reas, evolved from ideas explored in the Aesthetics and Computation Group at the MIT Media Lab. The system facilitates teaching many computer graphics and interaction techniques including vector/raster drawing, image processing, color models, mouse and keyboard events, network communication, and object-oriented programming (Reas and Fry, 2007). One of the stated aims of Processing is to act as a tool to get non-programmers started with programming, through the instant gratification of visual feedback. Processing it is primarily used by students, artists, designers, researchers, and hobbyists for learning, prototyping, and production.

The Processing language is simplification of the Java language. When programming in Processing the code is translated into pure Java before compiling.

Processing is distributed with the following set of libraries:

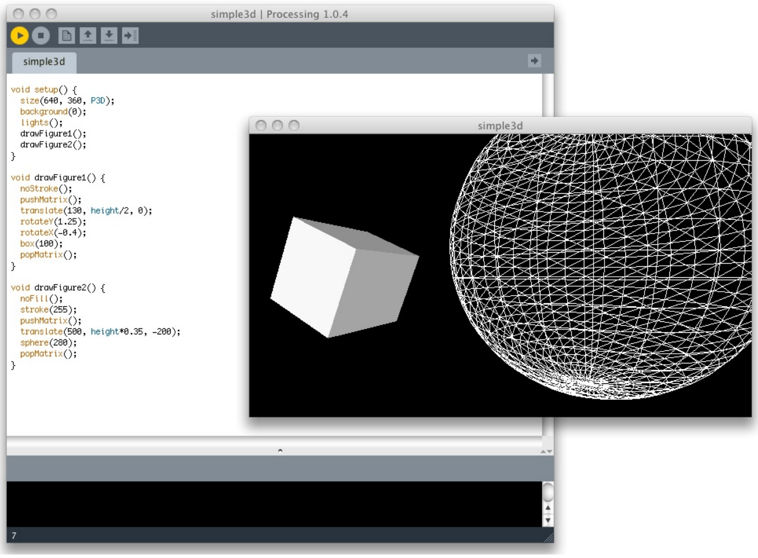
- Video Interface to Apple's QuickTime for using a camera, playing movie files, and creating movies.
- Network Sending and receiving data via the Internet through the creation of simple clients and servers.
- Serial Supports sending data between Processing and external hardware via serial communication (RS-232).
- PDF Export Generates PDF files.
- OpenGL Support for exporting OpenGL accelerated sketches.
- Minim Uses the JavaSound API to provide an easy-to-use audio library.
- DXF Export Lines and triangles from modes can be sent directly to a DXF file.
- Arduino Allows direct control of an Arduino board through Processing.
- Netscape.JavaScript Methods for interfacing between Javascript and Java Applets exported from Processing.

It is easy to extend Processing integrating existing Java libraries. The processing libraries encapsulate the Java libraries, simplifying their usage. There are many thirdy-parts libraries useful for creating interactive installation available from the Processing web site:

- bluetoothDesktop Send and receive data via Bluetooth wireless networks.
- EEML Library Extended Environments Markup Language (EEML) is a protocol for sharing sensor data between remote responsive environments.
- ezGestures A modular gesture recognition library.

- Most Pixels Ever Framework for spanning Processing sketches across multiple screens.
- OpenCV An OpenCV interface for processing including blob detection, face recognition.
- proMidi Allows Processing to send and receive midi signals.
- QRCode Reads QR Code images, a two-dimensional barcode format.
- RiTa An easy-to-use natural language library that provides simple tools for experimenting with generative (or computational, or digital) literature.
- TUIO Client library for the simple creation of tangible interactive surfaces, receiving TUIO data from object and multi-touch trackers such as reacTIVision.

Processing includes a *sketchbook*, a minimal Integrated Development Environment (IDE) for organizing projects. The IDE, shown in Figure 2.11, consists of a text editor for writing code, a message area, a text console, tabs for managing open files, and a toolbar with buttons for common actions. The console display compilation error message and text output by Processing program, the message area gives feedback while for several operations (e.g. saving, loading, compiling). When a Processing program starts, it open a new window for the graphical output.



```
void setup() {
  size(640, 360, P3D);
  background(0);
  lights();
  drawFigure1();
  drawFigure2();
}

void drawFigure1() {
  noStroke();
  pushMatrix();
  translate(130, height/2, 0);
  rotateY(1.25);
  rotateX(-0.4);
  box(100);
  popMatrix();
}

void drawFigure2() {
  noFill();
  stroke(255);
  pushMatrix();
  translate(500, height*0.35, -200);
  sphere(280);
  popMatrix();
}
```

Figure 2.11: *A screenshot of the Processing IDE, with a simple program placing 3D objects in space. The* lights() *method reveals their imagined dimension. The* box() *and* sphere() *methods each have one parameter which is used to specify their size. These shapes are positioned using the* translate() *function.*

Processing has spawned another project, Wiring, which uses the Processing IDE together with a simplified version of the C programming language as a way to teach artists how to program microcontrollers. There are now two separate hardware projects, Wiring and Arduino, using the Wiring environment and language.

2.2.2 Application Software for Lighting Control

In this section, we introduced two lighting control software as example of two different approaches for lighting control. The first one is a popular commercial software suite for creating and visualizing lighting show. We present this software as an example of the state of the art of the stage lighting solutions. The second is a computer–assisted lighting design and control system presented by Michael Sperber in his Ph.D. thesis.

2.2.2.1 Sunlite Suite

The Sunlite Suite[1] is a set of software for creating and visualizing lighting show. The two most significant softwares are Easy Show and Magic 3D Easy View.

Easy Show is a tool for synchronizing lighting effects with audio and video. Similar to audio editing software, it includes *timelines* where users can control their lighting effects. Lighting effects timelines can also be synchronized with audio and video timelines.

Magic 3D Easy View is a lighting visualizer software. It provides a real–time 3D rendering of a stage. As shown in Figure 2.12, it allows the lighting designer to preview light movement, colors, and every other effect available in robotic/intelligent lighting (e.g. strobe, dimmer, shutter, etc). It is also possible to insert objects to customize the stage (i.e. drums, piano, furniture) from a predefined libraries of objects, or to import objects from a CAD software. The software allows to reconstruct stages and venues in a very realistic way. It is possible to record videos of the lighting shows and take still pictures.

2.2.2.2 Lula

Lula (Sperber, 2001) is a system for computer-assisted stage lighting design and control developed by Michael Sperber. Its main improvement from existing lighting control systems is its modelling of the conceptual structure of a lighting design rather than its implementation.

A more faithful representation of the structure of a lighting design requires re–examining all basic design premises of existing systems, and has resulted in a complete redesign of the concept of the lighting control system.

[1]http://www.nicolaudie.com

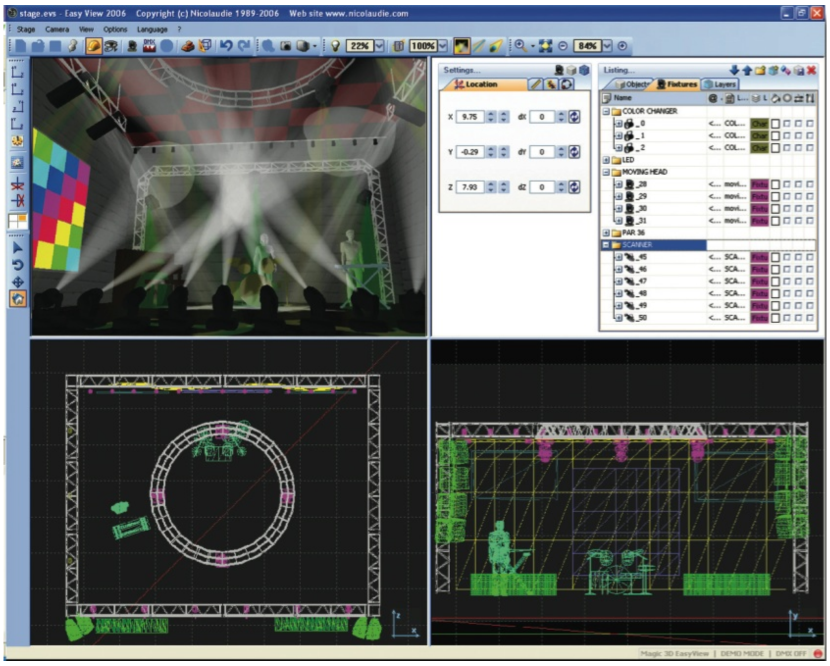
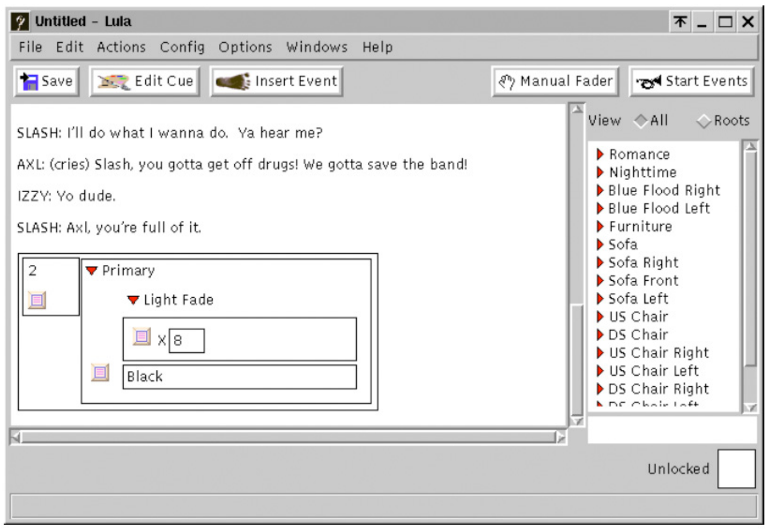

stage.evs - Easy View 2006 Copyright (c) Nicolaudie 1989-2006 Web site www.nicolaudie.com
Stage Camera View Options Language ?
Settings
Location
Listing
Object Fixtures Layers
COLOR CHANGER
LED
MOVING HEAD
PAR 36
SCANNER

Figure 2.12: *A screenshot of the Sunlite Magic 3D Easy View. In the top left quadrant there is a frame of the 3D rendering of the stage, bottom left there is the top view of the stage, bottom right the front view. In the top right quadrant is shown the list of the fixtures.*

Lula tries to address another shortcoming of existing systems: these systems exhibit significant non–linearities between the user–interface controls and the actual situation on stage. Lula lighting component model is based on a rigorous formal specification. This specification is the basis for both the Lula internal data representations and its graphical user interface. According to the author, the uniformity of the specification is not a guarantee, but a necessary prerequisite and good indicator for the usability of the interface.

Lula internally expresses all light changes as animations in term of Functional Reactive Programming (FRP) (Elliott and Hudak, 1997). FRP is a programming technique for representing values that change over time and react to events. For constructing complex animations, the user has direct access to FRP via a built-in *domain–specific programming language* called *Lulal*. Lulal is a higher-order, purely functional, strongly typed language. Lulal syntax is borrowed from Scheme language. Lulal is for the designer of dynamic lighting components, not for every user of a lighting control system. The language is hidden to the user who merely wants to assemble a show from cues, fades and prefabricated pieces. Such users can create their shows through the *script editor*, that allows pasting *events* into a theatrical script. An event corresponds to an event on stage which requires a coordinated lighting change. A screenshot of the script editor is shown in Figure 2.13.



SLASH: I'll do what I wanna do. Ya hear me?

AXL: (cries) Slash, you gotta get off drugs! We gotta save the band!

IZZY: Yo dude.

SLASH: Axl, you're full of it.

Figure 2.13: *A screenshot of the Lula Script editor showing show the final lighting event of a show. On the left there is the editor pane, on the right there is the list of the lighting component. (Source: Sperber, 2001, fig. 5, p. 131)*

In theatrical use, Lula drastically cuts down on the time usually needed for programming the control system. Lula is especially attractive for touring productions: since it allows separating the conceptual components of a design from its implementation, the operator can preserve large parts of the programming between venues.

2.2.3 Enabling Communications Technologies

Most automated lighting fixtures use a standard protocol. There are several protocols suitable for Adaptive Lighting. We divide the protocols in two families, according to the protocol designation. There are protocols designed for the Stage Lighting control and protocols for the Domotics.

The most common protocols of the two families are the following:

Stage Lighting Protocols

- Analog (0–10V) Control
- DMX512-A
- Art–Net II
- ACN
- MIDI
- Open Sound Control

Domotics Protocols

- X10
- INSTEON
- BACNet
- LonWorks
- European Home Systems Protocol (EHS)
- BatiBUS
- European Installation Bus (EIB)
- KNX
- SCS BUS – OpenWebNet

In the following paragraphs we describe in details the most significant of such communication protocols.

2.2.3.1 DMX512-A

DMX512-A is an unidirectional communications protocol used to control stage lighting and effects (e.g. fog machine). It was created in 1986 by the United States Institute for Theatre Technology (USITT) as a standard for controlling stage lighting and it was subsequent revised in 1990.

The communications standard covers digital multiplexed signals, provides up to 512 control channels per data link. Each of these channels was originally intended to control lights intensity and can assume an integer value between 0 and 255 (i.e. 8-bit number). The value 0 corresponds to the light being completely off while 255 corresponds to the light being fully on. More complex devices use adjacent channels to control different aspects of their behavior. For example, an RGB moving light can use 5 channel: 3 for the RGB color intensities, 1 for the pan (horizontal rotation) and 1 for the tilt (vertical rotation). To control position more accurately, some fixtures use 2 channels each for pan and tilt. This gives a 16-bit value range of 65536, permitting an higher accuracies for each axis.

According to the standard, a DMX512 controller is connected to the devices in a multi-drop bus topology commonly called a daisy chain. As shown in Figure 2.15, each device has an input and out connector. The controller is linked via a DMX512 cable to the input connector of the first device. A second cable then links output connector on the first device to the next device, and so on. The final output connector should have a terminating connector plugged into it.

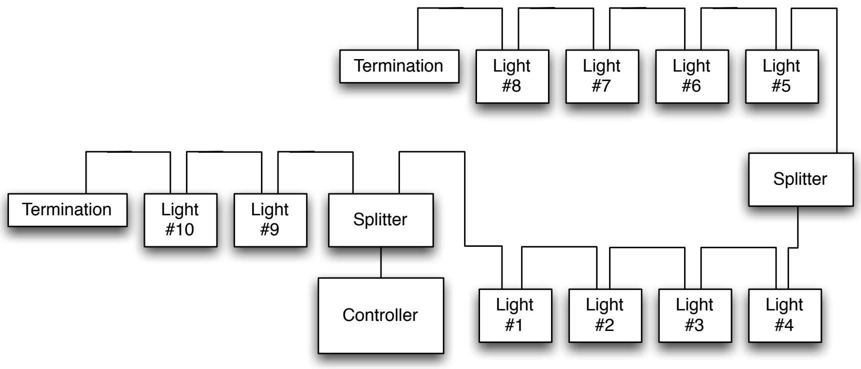


Figure 2.14: *An example of a DMX network with 10 lights and 2 splitters.*

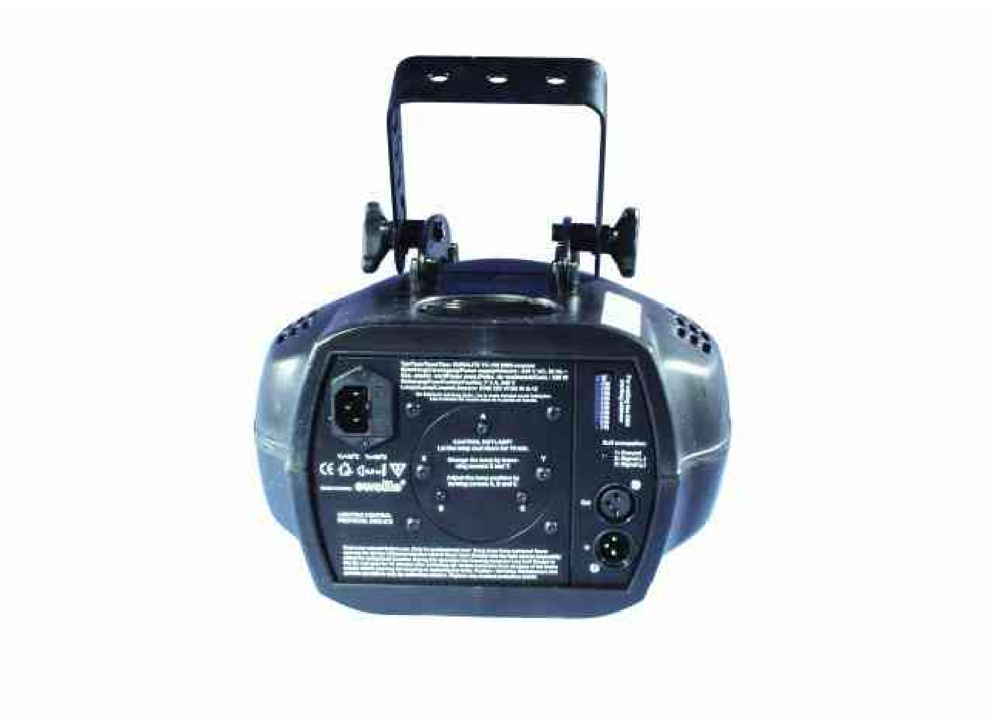


Figure 2.15: *The back of the EUROLITE TC-150 DMX Color changer spot. The two connectors on the bottom right of the picture are the DMX512 input and out connector. The DMX channel is configurable via dip switches on the top right.*

A DMX512 output from a DMX512 transmitter has the capacity to drive up to 32 units. In order to drive more than 32 units, a DMX Splitter is required. A Splitter consists of a DMX input connector and several DMX output connectors. Each of the DMX output connectors can now drive 32 units each. A Splitter is used also then the cabling between two fixtures is very long, since the signal can degrade significantly. An example of a DMX network is shown in Figure 2.14.

Many improvements to DMX512 have been proposed to address limitations such as the maximum slot count of 512 per universe, the unidirectional signal, and the lack of inherent error detection.

The 2004 revision of the standard also lays the foundation for the RDM (Remote Device Management) protocol through the definition of Enhanced Functionality. RDM allows for diagnostic feedback from fixtures to the controller by extending the DMX512 standard to encompass bidirectional communication between the lighting controller and lighting fixtures. RDM can be used for:

- Identification and classification of connected devices;
- Addressing of devices;
- Status reporting;
- Configuration.

DMX512 standard states that the data link shall utilize 5-pin XLR connector and 5 wire cables. However the DMX signal can being routed on other media such ethernet of wireless, in order to control distance or remote devices.

2.2.3.2 ACN

Architecture for Control Networks (ACN) is a suite of network protocols for theatrical control being developed by Entertainment Services and Technology Association (ESTA). ESTA started work on ACN in 1997, and the standard was finally released in 2006.

It may replace DMX as the control protocol for lighting systems and will be used for controlling more complex devices like media servers and audio mixers. ACN enables a number of components to be connected on a control network. ACN is platform and network independent, but, the most commonly used UDP/IP and will run over standard Ethernet and Wi-Fi networks. The layers of the ACN protocols are shown in Figure 2.16.

Every unit connected to an ACN network needs a unique address, known as Component Identifier (CID). CID is a 128-bit number, allowing for a large possible range.

The Device Description Language (DDL) is an XML language defining the device interface. Each devices is associated to a DDL document. The document describes the properties associated with behaviors, providing additional information about the property, such as what the property actually does, a name for the property to be used by the controller, and so on. Each properties may have several children. For example, for a moving light, the root property is the automated light itself, the pan function is a sub-property of the root, and the max and min degree of pan are sub-property

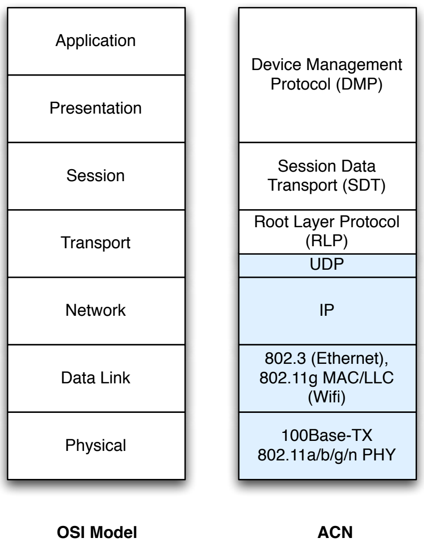

Application
Presentation
Session
Transport
Network
Data Link
Physical
Device Management Protocol (DMP)
Session Data Transport (SDT)
Root Layer Protocol (RLP)
UDP
IP
802.3 (Ethernet), 802.11g MAC/LLC (Wifi)
100Base-TX 802.11a/b/g/n PHY
OSI Model
ACN

Figure 2.16: *The layers of the Architecture for Control Networks (ACN), compared the OSI Model. The azure layers are not part of the ACN standard, but are the most commonly used.*

of pan. The Device Class Identifier (DCID) is used by a manufacturer to indicate a particular device model. Each device with the same DCIC is associated to the same DDL document.

The Device Management Protocol (DMP) is the mechanism to control, configure, or monitor specific properties in a connected device. Other functions of DMP include the handling of parameters that might be continuously updated, through the use of events and subscriptions.

The Session Data Transport (SDT) provides reliable multicasting and guarantees to layers above (e.g. DMP) that packets will be delivered to multiple receivers and and they arrive in the correct order.

The Root Layer Protocol (RLP) is the interface between ACN protocols and the lower-layer network transport protocols. RLP has been designed separately to ensure maximum network independence.

2.2.3.3 MIDI

The Musical Instrument Digital Interface (MIDI) is an standard protocol defined in 1982 that was originally designed for electronic musical instruments interconnection, and is now used widely in many parts of the entertainment industry. MIDI does not transmit an audio signal but only control information representing musical events is sent.

MIDI is a simple point-to-point interface, allowing devices to be connected in a simple master/slave relationship. Since the standard is unidirectional, a MIDI devices usually has both a receiver (MIDI In), a transmitter (MIDI Out). Sometimes, MIDI devices has a pass-through connector (MIDI Thru), that is used for daisy-chaining devices. The output of MIDI Thru is a copy of the data on the MIDI In.

An approach, better than daisy chain, for control applications is to use a MIDI Splitter, which takes one MIDI input and creates multiples copies of that input. With such a device, a hierarchical network can be created.

MIDI Show Control (MSC) is an open, standardized protocol for show control developed in 1991. The purpose of MIDI Show Control is to allow MIDI systems to communicate with and to control dedicated control equipment in theatrical and live performance. MSC enables several kinds of entertainment equipment to communicate with each other through the process of show control. Applications may range from a simple interface through which a single lighting controller can be instructed to start and stop, to complex communications with large, timed and synchronized systems utilizing many controllers of all types of performance technology.

MSC messages are transmitted in the same way as musical messages and are fully compatible with all conventional MIDI hardware. Commands are most often addressed to one device at a time. Each MSC message contains a device ID, determining to what address a message is intended. Since MIDI is a broadcast standard, all messages go to all devices. Each devices is responsible to check if it is the intended receiver of a particular message. There are 112 individual device IDs, 15 groups id (i.e. groups of devices to be addressed simultaneously) and a broadcast id, which is used to transmit global messages to all receivers in the network.

One of the limitation of MSC, is commands are completely open loop: no feedback or confirmation of any kind is required for the completion of any action. When a controller sends a message out, it has no idea if the target device even exists.

2.2.3.4 X10

X10 is a power–line based home automation protocol, developed in the 1975. The control signals are transmitted via existing power lines, without the need of dedicated buses. In addition to the power-line, X10 provides remote controls based on radio communication. An example of an X10 network is shown in Figure 2.18. X10 is used to trigger simple control events. However, it is not suitable for critical applications because no feedback channel is provided. The messages are modulated on a 120 kHz signal on the power line. The data rates are slow, about 20 bit/s.

X10 messages consist of a four bit *house code* followed by one or more four bit *unit code*, finally followed by a four bit command. Combining the *house code* and the *unit code*, 256 devices can be addressed and controlled in a X10 network. The X10 protocols defines the following 16 commands are the following:

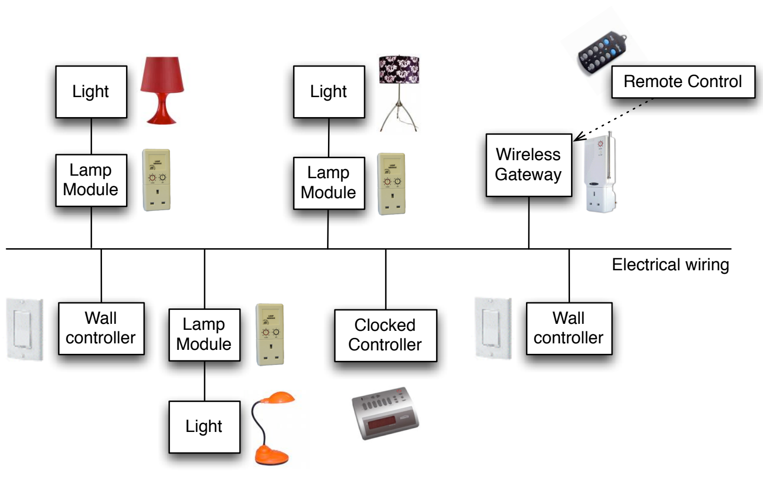


Figure 2.17: *An example of an X10 network comprising 3 lamp, 2 wall controller, a clocked controller a wireless gateway and a remote controller.*

Code	Command	Description
0000	All Units Off	Switch off all devices
0001	All Lights On	Switches on all lighting
0010	On	Switches on a device
0011	Off	Switches off a device
0100	Dim	Reduces the light intensity
0101	Bright	Increases the light intensity
0110	All Light Off	Switches off all lighting
0111	Extended Code	Commands extension
1000	Hail Request	Transmitted to see if there are any X10 transmitters
1001	Hail Acknowledge	Response to the Hail command
1010	Pre-Set Dim	Select the first predefined level of light intensity
1010	Pre-Set Dim	Select the second predefined level of light intensity
1100	Extend Data	Commands extension
1101	Status is On	Response indicating that the device is switched on
1110	Status is Off	Response indicating that the device is switched off
1111	Status Request	Request requiring the status of a device

2.2.3.5 KNX

KNX is a standardised network communications protocol for intelligent buildings. It is the successor of three previous systems for home and building automation: the European Home Systems Protocol (EHS), BatiBUS, and the European Installation Bus

(EIB). The standard is based on the communication stack of EIB.

A KNX installation consists of a set of devices connected into a network. The standard is designed to be independent of any particular communication media. The KNX system offers the choice for the manufacturers choose between several physical layers, or to combine them. The KNX messages can be delivered on powerline networking, twisted pair wiring, radio, infrared. KNX has also an unified service and integration solutions for IP-enabled media like Ethernet, Bluetooth, WiFi. On the different media, the transmission speeds are different. For example, the EIB-compatible mode on twisted pair wiring reaches a transmission speed of 9.6 kbits/s.

The KNX device on a network can be identified by their individual address, or by their unique serial number (similar to the MAC address in the Ethernet networks), depending on the configuration mode. The individual address is a 16-bit, allowing up to 65536 devices on a single network.

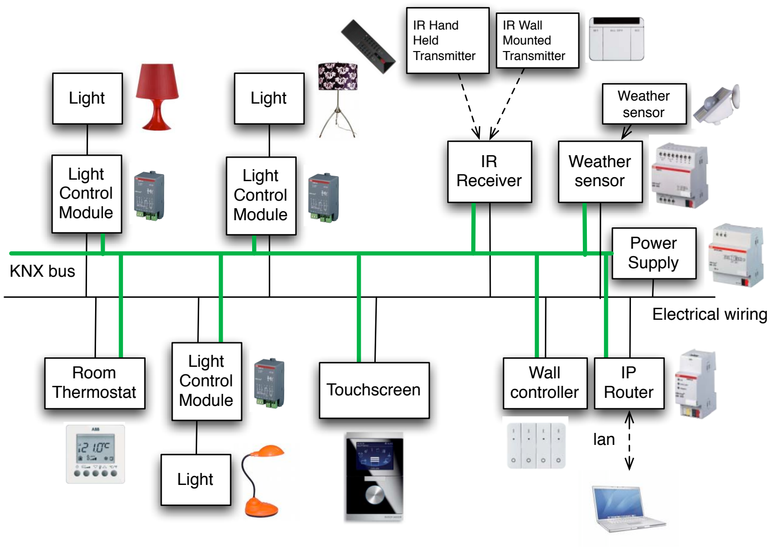


Figure 2.18: *An example of an KNX home network over twisted pair bus, comprising several devices.*

The sciences do not try to explain, they hardly even try to interpret, they mainly make models.

John Von Neumann

3 Cellular Automata And Other Cellular System

3.1 Cellular Automata

CELLULAR Automata (CA), introduced by John von Neumann as an environment for studying self-replicating systems (von Neumann, 1966), have been primary investigated as theoretical concept and as a method for simulation and modeling (Weimar, 1997b). CA are a class of spatially and temporally discrete mathematical systems characterized by local interactions (Wolfram, 1986b). Even if the interaction is based on simple local rules, the resulting structures from the CA evolution may be extremely complex (Wolfram, 1994, 1984a).

Informally, a cellular automaton is a regular array of identically programmed units called cells. Each cell is characterized by an internal state selected from a finite set of states. At discrete time step, each cell changes its state according to a finite set of prescribed rules for local transitions and the neighbors states.

3.1.1 Formal Definition

We call cellular automaton the 4-tuple $(\mathcal{L}, \mathcal{S}, \mathcal{N}, f)$ where

- $\mathcal{L}$ is a regular lattice,
- $\mathcal{S}$ is a finite set of states,
- $\mathcal{N}$ is a finite set of neighbors,
- $f : \mathcal{S}^n \rightarrow \mathcal{S}$ is a transaction function.

3.1.1.1 Regular Lattice

A d-dimensional lattice denoted by $\mathcal{L}$, consists of a periodic paving of a d-dimensional space domain. Every element of $\mathcal{L}$ is called cell. A cell is denoted by c, will be indexed by a tuple $(i_l, i_2, ..., i_u)$ of integers. The definition of a cellular automaton requires

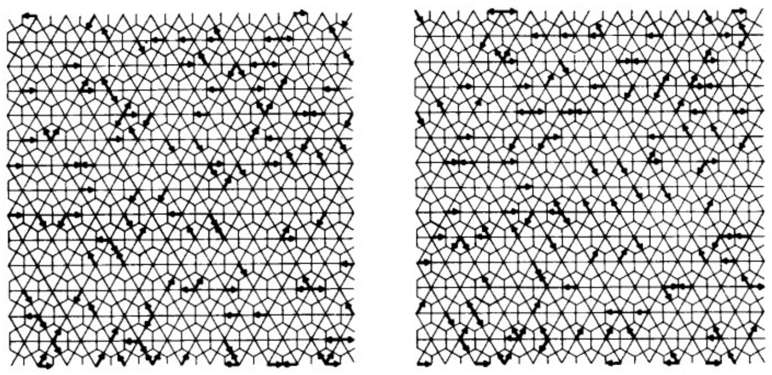


Figure 3.1: *Two successive microscopic configurations in a cellular automaton fluid model. Each arrow represents a discrete "particle" on a link of the hexagonal grid. (Source: Frisch et al., 1986, fig. 1, p. 1)*

the lattice to be regular, i.e., invariant with respect to translation in d independent directions.

We can consider various possibilities for one, two, and three dimensions. In the one-dimensional case (1D), we have a linear array of cells, that may be wrapped into a torus for periodic boundary conditions.

In two-dimensional (2D), there are three regular lattices depending on the cell shape, namely triangular, square, and hexagonal lattices (Weimar, 1997b). Triangular lattices can be useful in some cases because the small number of cells neighbors. Square lattices are simple to represent and visualize, but in some cases they have insufficient isotropy. Hexagonal lattices have a lower anisotropy compared to the triangular and square. Often this feature makes simulation appear more natural, and in some cases it is necessary to model the phenomena correctly (Wolfram, 1986a; Frisch et al., 1986). Figure 3.1 shows an example of an hexagonal cellular automaton fluid model.

In three dimensional (3D), there are many possible regular lattices, but the most common is the cubic lattice, since it is easiest to represent. According to Frisch et al. (1986), there are no regular three-dimensional lattices that have sufficient symmetry to correctly simulate hydrodynamics in the particle representation.

3.1.1.2 State set

The state set, denoted by $\mathcal{S}$, is a nonempty, finite and ordered set of state values. It may consist, for simplicity, of integer numbers. Cell states are given at discrete times $t = 0, 1, 2.....$ The state of cell c at time t is denoted by $s_t(c)$ and the state of its neighborhood by $s_t(N(c))$. Then we have for every $c \in \mathcal{L}, s_t(c) \in \mathcal{S}$, and $s_t(N(c)) \in \mathcal{S}^n$. $s_t(N(c))$ represents the neighborhood configuration at time t and $s_t(\mathcal{L})$ represents the configuration of the cellular automaton at time t.

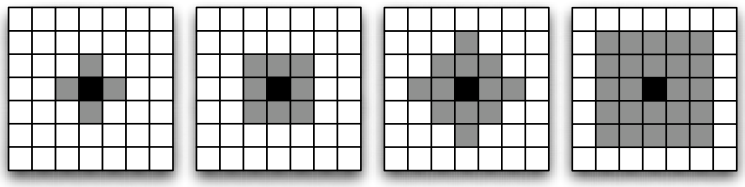


Figure 3.2: *Example of different neighborhoods in two-dimensional lattices: von Neumann, Moore, von Neumann with radius 2, Moore with radius 2.*

3.1.1.3 Neighborhood

By definition, a cellular automata rule is local, so the updating of a given cell requires one to know only the state of the cells in its *neighborhood*. We introduce a cell neighborhood as a set of cells which affect the evolution of a central cell. A neighborhood is then defined by the mapping

$$\mathcal{N} : \mathcal{L} \to \mathcal{L}^n \tag{3.1}$$

which makes a relation between the central cell c and n neighboring cells $c_1, c_2, ..., c_n$. We denoted by $\mathcal{N}(c)$ the set of neighbors of cell c. The integer n (or the number of neighbors) will characterize the size of the neighborhood.

In two-dimensional (2D) lattices, the neighborhoods are often considered: the von Neumann and the Moore neighborhood. Denoting the cell c at position (i, j) as $c_{i,j}$, the von Neumann neighborhood is defined as

$$\mathcal{N}_{c_{i,j}} = \{c_{k,l} \in \mathcal{L} : |k - i| + |l - j| \leq 1\} \tag{3.2}$$

and the Moore neighborhood is defined as

$$\mathcal{N}_{c_{i,j}} = \{c_{k,l} \in \mathcal{L} : |k - i| \leq 1, |l - j| \leq 1\} \tag{3.3}$$

The generalization of von Neumann neighborhood of radius r is defined as

$$\mathcal{N}_{c_{i,j}} = \{c_{k,l} \in \mathcal{L} : |k - i| + |l - j| \leq r\} \tag{3.4}$$

and the Moore neighborhood of radius r is defined as

$$\mathcal{N}_{c_{i,j}} = \{c_{k,l} \in \mathcal{L} : |k - i| \leq r, |l - j| \leq r\} \tag{3.5}$$

The mentioned neighborhoods are shown in Figure 3.2. The neighborhoods may be punctured, i.e., $c \notin \mathcal{N}(c)$, or may include the central cell, i.e., $c \in \mathcal{N}(c)$.

Another popular two-dimensional neighborhood is the Margolus (Margolus, 1984), which consists of partitioning the space into adjacent blocks of 2×2 cells in which the automata's rule is applied completely locally. The two partitioning, called odd and even, are possible (e.g $< c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1} >$ or $< c_{1,1}, c_{1,2}, c_{2,1}, c_{2,2} >$). The neighborhood alternates between these two situations at even and odd time steps.

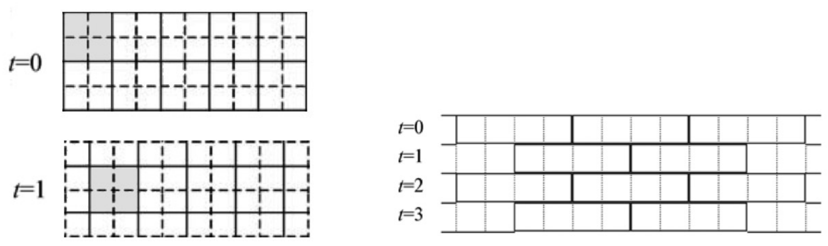


Figure 3.3: *On the left, 2D Margolus neighborhood: even (solid lines) and odd (dotted lines) partitions of a two-dimensional array into 2x2 blocks. One block in each partition is shaded. On the right 1D version of the Margolus neighborhood. The partitions alternate between even and odd steps. Solid lines delimit a partition. (Source: Cerdá et al., 2005, fig. 2, p. 281)*

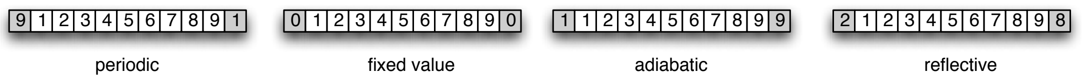


Figure 3.4: *Various types of boundary conditions on a 1D cellular automata. The shaded blocks represent the virtual cells added at the extremities to complete the neighborhoods.*

It is possible to generalize Margolus neighborhood to arbitrary dimensions and arbitrary block sizes. A 1D Margolus neighborhood is presented in Cerdá et al. (2005), the Necker neighborhood is an extension of the Margolus neighborhood for the 3D lattices. Figure 3.3 shown two examples of one and two dimensional Margolus neighborhoods.

3.1.1.4 Boundary Conditions

There are mainly two reasons to define Cellular Automata on finite lattices. The first is that in practice, it is impossible to simulate a truly infinite lattice on a computer. We can simulate a CA over a infinite lattice only if the active region always remain finite. The other reason are the natural boundary of the phenomenon we want to simulate. For example, in the simulation the coffee percolation process (Bandini et al., 1992), the boundary conditions are determined by the shape of the percolation device (i.e. the coffee machine filter). Event if the phenomena has natural boundaries, it not always necessary to impose boundaries conditions to the automaton: another possibility would be to design transactions functions depending on the number of neighborhood.

There are four kinds of boundaries we will consider here (shown in Figure 3.4):

- *Periodic Boundaries*, obtained by periodically extending the lattice, are a very common solution. That is one supposes that the lattice is embedded in a torus-like topology. No cell in this topology "see" the boundary in any way and thus it come closes to simulating an infinite lattice, and are therefore often used (Weimar, 1997b).

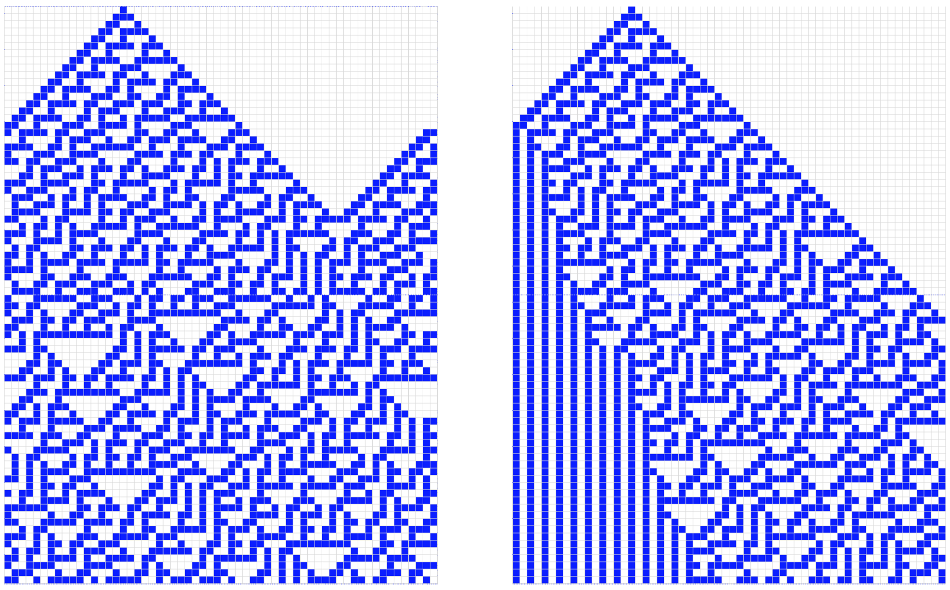


Figure 3.5: *On the left, the evolution of the Elementary Cellular Automaton Rule 30 with periodic boundaries, on the right, the same automaton with fixed value boundaries.*

- *Fixed Value Boundaries* are defined so that the neighborhood is completed with cell having a pre-defined fixed value.
- *Adiabatic Boundaries* are similar to a fixed value Boundaries, but the value dynamically change according to the value of the boundary cells.
- *Reflective Boundaries* are obtained by reflecting lattices at boundaries.

As shown in Figure 3.5, the choice of the boundary condition can heavily influenced the dynamic evolution of an automaton. The figure shown two evolutions of the *Rule 30* Elementary Cellular Automaton with different boundaries conditions. On the left is presented the evolution of the automaton with periodic boundaries, displaying aperiodic, chaotic behavior. On the right is shown the same automaton with zero as fixed boundary value. The evolution of the automaton leads to an homogeneous configuration.

3.1.1.5 Transition Function

The transition function governs the evolution of the system itself. It may be given by an analytical function, a matrix, or a set of transition rules. The transition function f may be considered as a mapping $\mathcal{S}^n \to \mathcal{S}$ given by

$$f : \mathcal{S}^n \to \mathcal{S} \tag{3.6a}$$

$$s_{\mathcal{N}(c)}^{(t)} \to s_c^{(t+1)} \tag{3.6b}$$

where $s^t_{\mathcal{N}(c)}$ are the states of cells in the neighborhood $\mathcal{N}(c)$ at time t and $s_c^{(t+l)}$ is the state of the cell c at time $t+1$. The relation

$$s_c^{(t+l)} = f(s^{(t)}_{\mathcal{N}(c)}) \tag{3.7}$$

is the state equation of the cellular automaton.

The evolution of a cellular automaton can be described by a table specifying the state a given cell will have in the next generation based on the value of the cell itself and the value of the neighbors cells. The table size, for an automaton with s states and n neighbors, is s^n.

3.1.2 Applications

Several researchers from diverse fields have identified cellular automata dynamics with problems in their own fields. The broad area of application of different techniques, technologies and approaches are presented one by one in the following paragraphs. This section does not represent an exhaustive description of all the aspects of this area but will provide the reader with a basic knowledge.

3.1.2.1 Games

The *Game of Life*, proposed in 1970 by the mathematician John Conway, is probably the most popular cellular automaton game. He imagined a two dimensional square lattice (with punctured Moore neighborhood) in which each cell can be in either death (state 0) or alive (state 1). The update rule is as follows:

- A dead cell with exactly three living neighbors becomes alife,
- A living cell with two or three living neighbors stays alive,
- In any other case, a cell dies or remains dead (overcrowding or loneliness).

From a formal point of view, the Game of Live is a Totalistic Cellular Automata, since the value of a cell at time $t+1$ depends only on the sum of the values of the cells in its neighborhood at time t. A cellular automaton is called *totalistic* if the value of a cell depends only on the sum of the values of its neighbors at the previous time step, and not on their individual values (Wolfram, 1983b)

We call $n_c^{(t)}$ the sum of the values of the neighbors of the cell c at time t.

$$n_c^{(t)} = \sum_{i \in \mathcal{N}(c)} s_i^{(t)} \tag{3.8}$$

We can write the Game of Life rule as

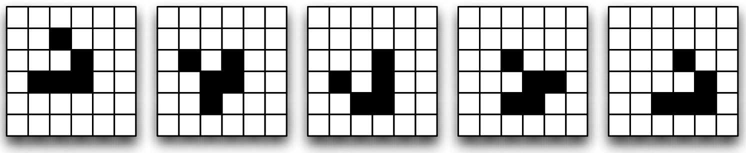


Figure 3.6: *An example of a simple* glider*, that reappears after 4 generations in the same orientation but in a different position.*

$$s_c^{(t+1)} = \begin{cases} 1 & \text{if} \quad n_c^{(t)} = 3, s_c^{(t)} = 0 \\ 1 & \text{if} \quad 2 \leq n_c^{(t)} \leq 3, s_c^{(t)} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

According to (Gardner, 1970), Conway chose the rules carefully, after a long period of experimentation, to meet three desiderata:

- There should be no initial pattern for which there is a simple proof that the population can grow without limit.
- There should be initial patterns that apparently do grow without limit.
- There should be simple initial patterns that grow and change for a considerable period of time before coming to end in three possible ways: fading away completely, settling into a stable configuration, or entering an oscillating phase.

In spite of the simplicity of these rules, surprisingly complex patterns can emerge. For example a *gun*, a configuration that repeatedly shoots out moving objects such as the *glider* (a configuration that moves) or a *puffer train* (a configuration that moves and leaves behind a trail of *smoke*). An example of a simple *glider* is shown in figure 3.6 Many more patterns have in investigated in Berlekamp et al. (1982) and Gardner (1983).

Besides the Game of Life, there are other games which have been modeled through cellular automata. Notable among these are the games which provide insights into the synchronization problems. The most popular, known as the *Firing Squad Synchronization Problem*, was introduced by Myhill in 1957.

The problem is design a transition function such that after some steps all cells go into a special "firing" state, that never occurred before. A valid solution must not depend on the length of the line: the same transaction function must be used for every possible finite line. That problem has been long studied and a very rich set of solutions has grown up (Moore, 1964; Waksman, 1966; Balzer, 1967; Kobayashi, 1977; Mazoyer, 1987; Yunes, 1994; Imai and Morita, 1996; Settle and Simon, 2002; Umeo and Yanagihara, 2009; Umeo et al., 2009).

3.1.2.2 Physical and Biological Systems

In physics, the time evolution of quantities is often governed by nonlinear partial differential equations, and the solutions of these dynamical system can be very complex. Cellular Automata are an alternative approach to differential equations in modeling laws of physics (Toffoli, 1984; Omohundro, 1984).

Good theoretical overviews and insights highlighting the possibilities of Cellular Automata replacing partial differential equations can be found in Ruffo and Lio (2001); Chopard and Droz (1998); Deutsch and Dormann (2008). This has resulted in investigation of CA models for pattern formation in reaction-diffusion systems (Madore and Freedman, 1983; Oono and Kohmoto, 1985; Winfree et al., 1985). There are two main approach to simulate nonlinear reaction-diffusion system with cellular automata: *Reactive Lattice Gas Automata* realize diffusion through a particles random walk over a lattice (Dab et al., 1990; Boon et al., 1995) and a second class, based on local average, that is a more macroscopic approach and is more efficient (Weimar, 1997a).

Wave propagation models have been studied by researchers based on Cellular Automata (Frisch et al., 1986; Chen et al., 1988; Cole et al., 1993). Chopard et al. (1997) had modeled wave propagation by Lattice Boltzmann approach applicable for practical situations such as the radio wave transmission in complex urban environments. In Komatsuzaki et al. (1999) is presented an acoustic wave propagation model for simulating sound source movement, sound diffraction by the presence of barriers and reflection due to inhomogeneity of acoustic media. Komatsuzaki and Iwata simulated the acoustic wave propagation for understanding fundamental sound absorption mechanism of porous materials and evaluating sound absorption performance, where the details of porous material structure is considered in the model (Komatsuzaki and Iwata, 2006).

Lattice Gas Automata are an important class of cellular automata to simulate fluid flows, that obey to the Navier-Strokes equations of hydrodynamics. LGA is based on a microscopic representation of particles that interact together, according to the laws governing the conservation of mass, momentum and energy. Many variant of the LGA has been developed, starting from a first square-lattice model (Hardy et al., 1976) to hexagonal-lattice models that better exhibit a fluid-like behavior (Frisch et al., 1986).

CA have also been employed ecological research, in order to study the population dynamics. Mingfeng He and Qui-Hui Pan and Shuang Wang proposed a cellular automata model containing movable wolves, sheep and reproducible grass. The authors introduced the energy rule and the predator-prey mechanism for wolf and sheep. With considering age-structured, genetic strings, minimum reproduction age, cycle of the reproduction, number of offspring, there are three possible states of a predatorprey system: the coexisting one with predators and prey, the absorbing one with prey only, and the empty one where no animal survived (He et al., 2005).

Mynett and Chen (2004) simulated the competitive growth and succession of two plant in an aquatic ecosystem. In order to model aquatic ecosystems, the authors

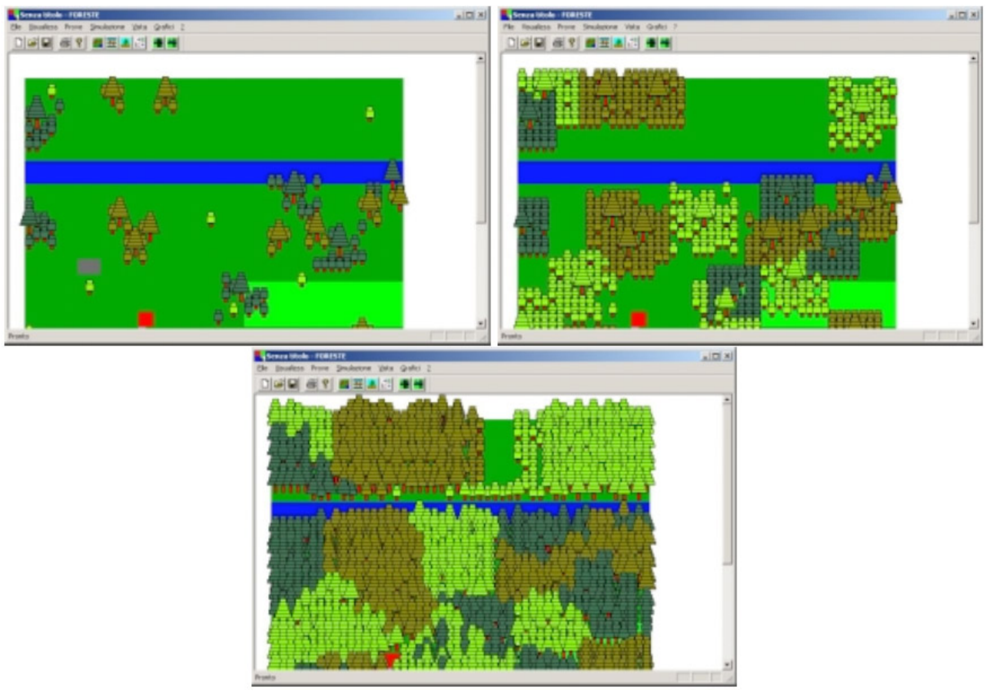


Figure 3.7: *An example of three different stages of the simulation of the evolution of a vegetable population composed by black locusts, oaks, and pine trees. (Source: Bandini and Pavesi, 2002, fig. 2, p. 208)*

extended the purely geometrically-determined rules to include external factors.

Bandini and Pavesi presented a model based on a two-dimensional Cellular Automata, that allows to model and simulate the evolution of heterogeneous vegetable populations composed by different perennial species (Bandini and Pavesi, 2002). In this model, the evolution of a vegetable population is influenced by the available resources (i.e. sunlight, water, substances present in the soil), and the different individuals compete for them. An example of three different stages of the simulation of the evolution of a vegetable population composed by black locusts, oaks, and pine trees, is shown in Figure 3.7.

Boer and Hogeweg (1992) developed a model of the immune system using an asynchronous cellular automaton.

Resnick (1997) has developed a chemotaxis CA model used to simulate random walk in response to a chemical gradient. The effect of random walk by single and a number of individuals in a system has been modeled and extensively studied in Shlesinger (1992).

3.1.2.3 Social Science

Several social scientists have been try to apply the methods of dynamical system analysis to social systems. In carrying out this project, however, one immediately runs into the fact that social systems comprise the interactions of conscious human beings, unlike physical systems in which the behavior of the elements arises only from their physical properties (Albin and Foley, 1999).

One early prototype of Cellular Automata model for the social science (without explicitly referring to Cellular Automata) was proposed in Sakoda (1971). The central goal of his model, already present in his unpublished dissertation of 1949, is to understand group formation. Another early example is the Schelling spatial proximity model (Schelling, 1971, 1969), that analyzed racial segregation processes among individuals belonging to two different groups. As shown in Figure 3.8, this model illustrates how distinctive patterns of spatial segregation can emerge even if individuals are only weakly segregationist.

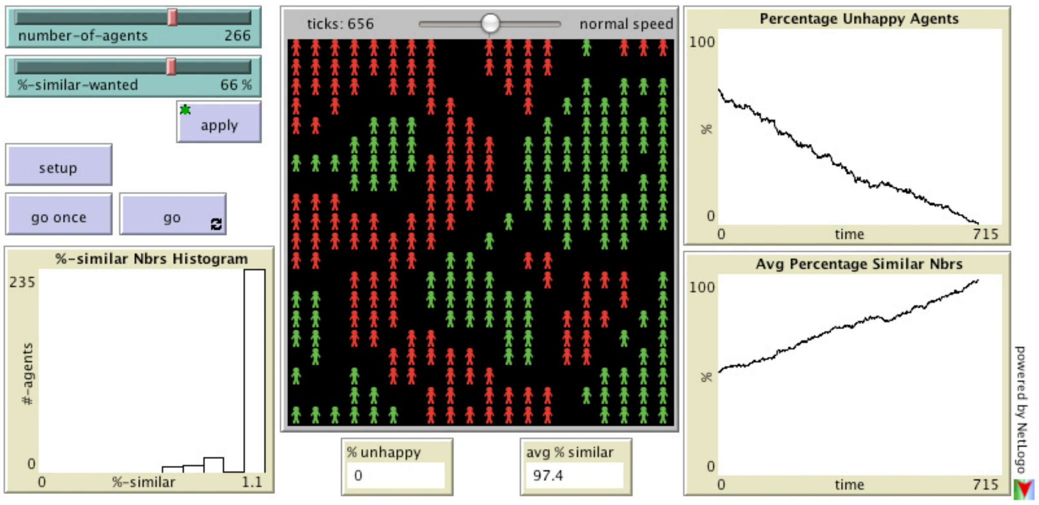


Figure 3.8: *Screenshot of a Luis R. Izquierdo implementation of the Schelling spatial proximity model of segregation.*

The first work that explicitly classified checkerboard models under cellular automata framework was proposed by Albin in his book "The Analysis of Complex Socioeconomic Systems" (Albin, 1975). In his work, the author emphasize the potential of Cellular Automata for understanding social dynamics.

Keenan and O'Brien (1993) proposed a one-dimensional Cellular Automata economic model to analyze pricing in a spatial setting. In his book "The Evolution of Cooperation" (Axelrod, 1984), Axelrod analyzed the cooperation dynamics within a Cellular Automata framework. Other studies of the dynamics of cooperation using Cellular Automata are presented in Nowak and May (1992, 1993); Kirchkamp (1995);

Messick and Liebrand (1995); Liebrand and Messick (1996).

A good analysis of Cellular Automata modeling for social science is presented in Hegselmann and Flache (1998).

3.1.2.4 Traffic flow

The main challenge in traffic flow modeling is the construction of macroscopic and microscopic models that lend themselves to a faithful representation of road traffic. In recent years, there is a significant use of cellular automata for modeling and simulating traffic flow (Nagel and Schreckenberg, 1992; Chopard et al., 1996; Nagel, 1996, 2002; Knospe et al., 2004; Campari et al., 2004; Maerivoet and De Moor, 2005). The first proposed CA traffic model, describing a single-lane traffic flow on a ring, is Nagel and Schreckenberg (1992).

The model is based on a one dimensional stochastic cellular automaton where the lattice represent a circular lane and each cells is either occupied by exactly one car (with speed $\in N_0, v \leq v_{max}$) or empty. At each step, the following four consecutive steps are performed in parallel for all the vehicles:

- Acceleration: if the velocity v is lower than v_max and if the distance d to the next vehicle is larger than $v + 1$, the speed is increased by one,
- Slowing down: if the distance to the next vehicle is less than the current speed ($d < v$), the speed is reduced to d,
- Randomization: with probability p, the velocity of a moving ($v > 0$) vehicle is reduced by one,
- Car motion: each vehicle proceeds by the value of its velocity (v).

According to the authors, the probabilistic step 3 (Randomization) is essential in simulation realistic traffic flow, since otherwise the dynamics is completely deterministic. It takes into account velocity fluctuations due to human behavior or external conditions. Without this step, every initial configurations of vehicles reaches a stationary pattern.

Figure 3.9 shown simulated traffic. Each line shows the traffic lane after one further complete velocity-update and just before car motion. Empty sites are represented by a dot, sites that are occupied by a car are represented by the value v (the vehicle velocity). The vehicles move from left to right. Note the backward motion of traffic jam (recognizable by the presence of vehicles with speed equals to zero).

3.1.2.5 Pedestrian and Crowd Dynamics

Different modeling techniques have been adopted to represent pedestrian and crowd dynamics, for example, forcebased models, Cellular Automata models and Multi Agent System models.

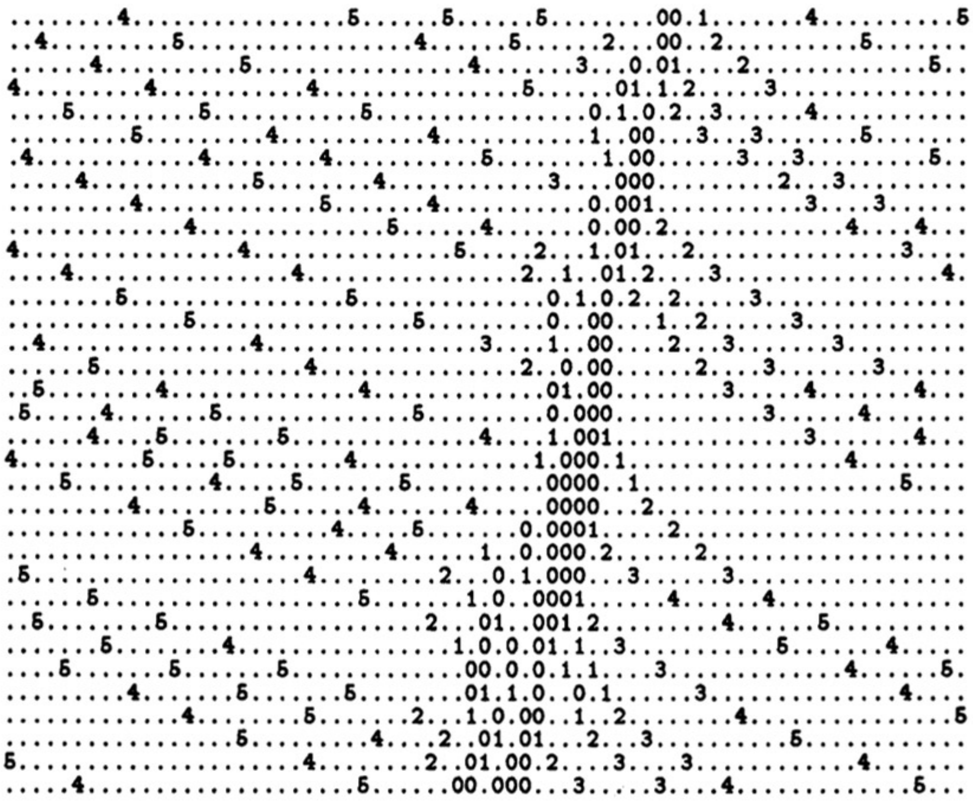


Figure 3.9: *Nagel-Schreckenberg simulated traffic. (Source: Nagel and Schreckenberg, 1992, fig. 2, p. 2224)*

Into the context of pedestrian and crowd dynamics, approaches based on Cellular Automata are demonstrated to be particularly adequate (Blue and Adler, 2001). The success of CA-based approaches derives mainly from the fact that are simpler to understand and to use by experts of several application contexts. Peculiarities of CA-based models is the explicit representation of the space as a regular grid (Schadschneider, 2002; Burstedde et al., 2002; Schreckenberg and Sharma, 2002), where the size of each cell is the minimal space occupied by a person. In this models, the state of cell includes the representation of the presence of individuals, of environmental obstacles and the direction of pedestrian's goals, through static and dynamic potential fields. The cell local interaction involved in transition function, therefore, represents also pedestrian movement by cell state change (e.g. an occupied cell becomes empty and, synchronously, an adjacent empty cell becomes occupied).

The *bi-directional model* (Blue and Adler, 2000, 1998, 1999a,b), introduced by Blue and Adler, focuses on the phenomenon of the crosswalk of two groups of pedestrians from a sidewalk to another. Each time step pedestrians decide to remain into its own queue or to shift in a neighbor one (the pedestrian chooses the most free queue). After the choice of the queue, pedestrians choose randomly its own speed. Conflict management can force a pedestrian to reduce the chosen speed or to remain at the same place, if there is no space to move. This model is very simple and very fast in simulation with a large number of pedestrians. The model allows to observe auto-organization phenomena as, for example, line formations, as shown in Figure 3.10.

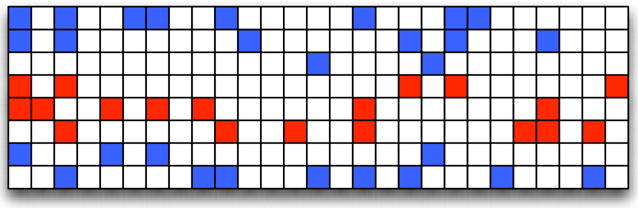


Figure 3.10: *Example of multiple lines formation in the* bi-directional model. *Red cells represent eastbound pedestrian, blue cells represent westbound pedestrian, and white cells are void.*

This *floor field model* was proposed in Burstedde et al. (2001). The aim of the model is to represent in a CA-based approach the attraction of pedestrians towards the target, and in the same time the attractions that pedestrians seem to have each other in some particular situations (i.e. panic scenarios).

The lattice is constituted by multiple layers. Each level contain different information about the same space. One level contains information about the current position and direction of each pedestrian. A second level give information about a sort of field of attraction forces discretized on the space and representing the force of attraction in each position of a given pedestrian towards the target. In the third level we can find the virtual wake traced by the pedestrian in its movement. Each pedestrian chooses the the next cell in which to move evaluating the intensity of the field attraction gradient presents in the second level.

3.1.2.6 Other Applications

In this section, we briefly presents some other CA applications, not limited to the previously presented categories.

Cellular automata can be applied in digital image processing. In Popovici and Popovici (2002) the authors discussed the application of two-dimensional cellular automata to the problems of noise removal and border detection in digital images. An attempt to use cellular automata in pattern recognition is proposed in Maji et al. (2002).The pattern recognition approach is designed around a general class of CA known as Generalized Multiple Attractor Cellular Automata (GMACA). GMACA employs non-linear CA rules with attractor cycle length greater than or equal to 1. A GMCA can efficiently model an associative memory (Ganguly et al., 2001, 2002). The Cellular Automata Machine is synthesized around a GMACA. The synthesis of GMACA can be viewed as the training phase of Cellular Automata Machine for pattern recognition. The result of the synthesis is the rule vector of the desired GMCA.

Toguchi, Akamine, and Endo presented a research on the generation of sound effects using a CA (Toguchi et al., 2008). According to the authors, the most commonly-used production methods for sound effects are: to record real sound, edit and then use it, to generate sound effects using a synthesizer,and to process recorded sound similar to the desired sound, or to use prerecorded sample collections for sound effects. The purpose of their research is development of a CA-based tool generating sound effects through physical simulation. There are a number of different types of sound effect that can be simulated with the proposed approach. For example, impact sound, aerodynamic sound, explosion sound, and friction sound.

In Xu et al. (2004), the authors proposed a cellular automata modeling an Elevator Group Control Systems (EGCS). The EGCS model consists of three modules: group control module, elevator movement module and assignment sequence module. Elevator movement module simulates the moving behaviors of elevators by using cellular automata. Assignment sequence module is established to logically store call orders to every elevator. According to the authors, computer simulations show that the proposed modeling method is creditable and the established EGCS model gives out satisfied performance.

CA have also been employed to control Modular Self-Reconfigurable Robot (MSR). MSR consist of many of identical and independent module. Each module is independent and include sensors, actuators, processor, communication and power. A module can connect and detach autonomously from the adjacent modules. An example of the movements of this kind of robots is shown in Figure 3.11. In Wu et al. (2005) the authors proposed a CA-based emergent control model. The approach comprise a two layers neural network with 7 inputs and single output, simulating the nonlinear rules function. The feature vector of module is the input of CA rules and the action of module is output of CA. The simulation results show that emergent control based CA is great significance to enhance robustness and scale extensibility of MSR.

3.1.3 Elementary Cellular Automata

The simplest, non-trivial, class of 1D cellular automata are the Elementary Cellular Automata (ECA) that have two possible values for each cell (0 or 1), and rules that depend only on nearest neighbor values (Wolfram, 1982).

The evolution of an elementary cellular automaton can completely be described by a table specifying the state a given cell will have in the next generation based on the value of the cell to its left, the value the cell itself, and the value of the cell to its right (Wolfram, 1982). Since each transaction table can be represented with 8 binary states, there are a total of 256 elementary cellular automata and each transaction function can be indexed with an 8-bit binary number. Figure 3.12 shown an example of the rule 30. Each of the eight possible sets of values for a cell and its nearest neighbors

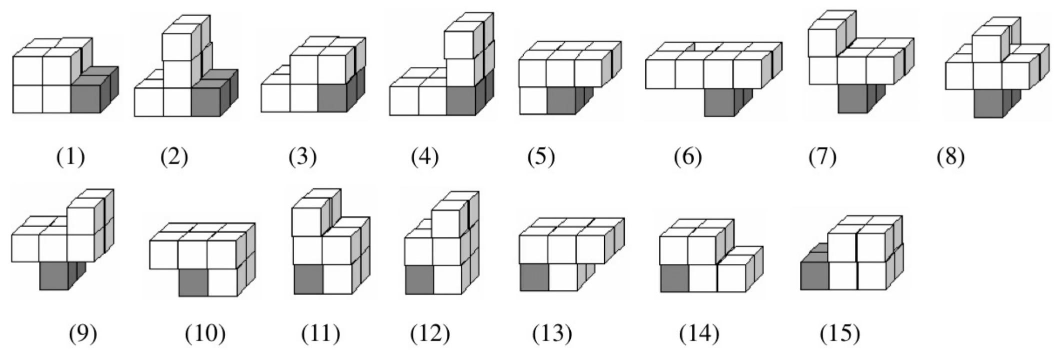
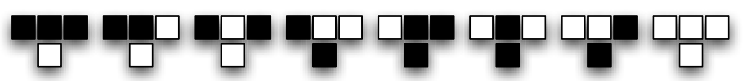


Figure 3.11: *Example of the movement of a Modular Self-Reconfigurable Robot composed by 4 modules (represented by the white cubes) in front of an obstacle (represented by the grey cubes). (Source: Wu et al., 2005, fig. 7, p. 186)*

appear on the upper line, while the lower line gives the value to be taken by the cell on the next time step.



0 0 0 1 1 1 1 0

$0*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 30$

Figure 3.12: *Example of the transition table for the elementary cellular automaton rule 30. Each of the eight possible sets of values for a cell and its nearest neighbors appear on the upper line, while the lower line gives the value to be taken by the cell on the next time step.*

The evolution of an elementary cellular automata and, more in general, one-dimensional cellular automat, can be displayed with ease, as shown in Figure 3.13. The evolution of the automaton is illustrated by starting with the initial state (generation zero) in the first row, the first generation on the second row, and so on.

An important aspect in theory of Elementary Cellular Automata is classification, dividing cellular automata into groups with related properties. More than one rule space classification scheme has been used in the literature. The first qualitative behavior classification, dividing cellular automata according to their observed periodic or chaotic behavior, was proposed by Wolfram (Wolfram, 1984b). The classification proposed by Wolfram in divides the automata into 4 classes:

Class 1 cellular automata evolve after a finite number of steps to a unique homogeneous state, in which all sites have the same value, from almost all initial configurations. Such cellular automata may be considered to evolve to simple "limit points" in phase space; their evolution completely destroys any information on the initial state.

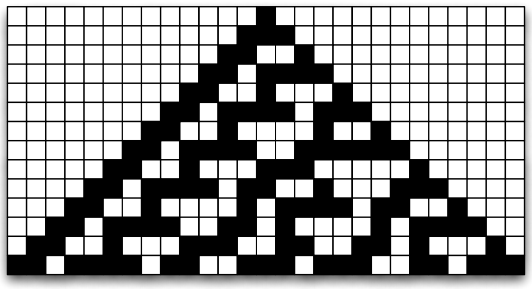


Figure 3.13: *Example of the evolution of a one-dimensional cellular automaton. The evolution is illustrated by starting with the initial state in the first row, the next generation on the second row, and so on. The figure above illustrated the first 13 generations of the rule 30 elementary cellular automaton, starting with a single black cell.*

Class 2 evolution leads to a set of separated simple or periodic structures. Cellular automata generate separated simple structures from particular (typically short) initial site value sequences.

Class 3 evolution leads to a "chaotic" patterns, from almost all possible initial configurations. After sufficiently many time steps, the statistical properties of these patterns are typically the same for almost all initial states. In particular, the density of nonzero sites typically tends to a fixed nonzero value.

Class 4 evolution leads to complex localized structures sometimes long-lived.

This classification was very important since it motivated numerous researches but was at the same time criticized by many authors on grounds that no formal definitions of the classes were given (Fatès, 2003).

Later on, Li and Packard proposed a series of refinements in the original Wolfram classification (Li, 1992; Li and Packard, 1990; Li et al., 1990b). The following is one version (Li, 1992; Oliveira et al., 2001) of their classification scheme, which divides the rule space into the following six classes:

Null rules : the limiting configuration (i.e., the automaton state of the entire lattice) is formed only by zeros or ones.

Fixed-Point rules : the limiting configuration is invariant, with possibly a spatial shift, by applying the cellular automaton rule, excluding all zeros or all ones configurations.

Two-Cycle rules : the limiting configuration is invariant, with possibly a spatial shift, by applying the rule twice.

Periodic rules : the limiting configuration is invariant by applying the automaton rule L times, with the cycle length L either independent or weakly dependent on the system size.

Edge of Chaos rules : although their limiting dynamics may be periodic, the convergence time can be extremely long and they typically increase more than linearly with the system size. One hallmark of this class of rules is its marginal stability with respect to perturbations, and another is its poor convergence of any statistical property such as the transient time.

Chaotic rules : they produce non-periodic dynamics. These rules are characterized by the exponential divergence of its cycle length with the system size, and for the instability with respect to perturbations. The transient time can either be long or short.

In Fatès (2003) the author proposed a classification based on the observation of the statistical evolution of the density, i.e. the number of cells in state one divided by the total number of cells. If the density evolves with large statistical distributions, the automaton is classified as chaotic-looking. If the evolution of the density eventually enters into a cycle of small length, the automaton is classified as periodic-looking.

3.1.4 Stochastic Cellular Automata

Stochastic Cellular Automata are an extension of the cellular automat in which the transition function has a probabilistic component. There is a considerable amount of freedom in the way to randomize the transaction rule. For instance, Burks (1970) chooses at random the rules that are applied in the updates. Buvel and Ingerson (1984) allow, for each cell, a certain probability that the respective cell is not updated. Lee et al. (1990) introduce the adaptive Stochastic Cellular Automata in which the probabilistic rules are nonuniform. For a broad overview of stochastic Cellular Automata we refer to (Wolfram, 1986c). One popular Stochastic Cellular Automata model is introduces in Nagel and Schreckenberg (1992), describing single-lane traffic flow on a ring.

3.1.5 Asynchronous Cellular Automata

Cellular Automata have traditionally treated time as discrete and state updates as occurring synchronously and in parallel. The state of every cell of the automaton is updated together, before any of the new states influence other cells. The synchronous approach assumes the presence of a global clock to ensure all cells are updated together.

Several authors (e.g. Paolo, 2000; Thomas and Organization., 1979) have argued that asynchronous models are viable alternatives to synchronous models and suggest that asynchronous models should be preferred where there is no evidence of a global clock. Nehaniv (2003) has demonstrated an asynchronous CA model that can behave as a synchronous CA, due to the addition of extra constraints on the order of updating.

Cornforth, Green, and Newth argue that asynchronous updating is widespread and ubiquitous in both natural and artificial networks (Cornforth et al., 2005). They identified two classes of asynchronous behavior: Random Asynchronous (RAS), and Ordered Asynchronous (OAS) updating. Random Asynchronous includes any process in which at any given time individuals to be updated are selected at random according to some probability distribution; Ordered Asynchronous includes any process in which the updating of individual states follows a systematic pattern. The authors presents a total of six specific update pattern (shown in Figure 3.14:

- Synchronous: all cells are updated in parallel at each time step,
- RAS - Random Independent: at each time step, a cell to update is chosen at random,
- RAS - Random Order: at each time step, all nodes are updated, but in random order (each node is updated exactly once per time step),
- OAS - Cyclic: at each time step a cell is chosen according to a fixed update order, which was decided at random during initialization phase,
- OAS - Clocked: a timer is associated to each cell, so that updating is autonomous and proceeds at different rates for different cells,
- OAS - Self-sync: similar to the clocked, but incorporates local synchrony.

In order to investigate the differences in behavior that result from the updating schemes described above, in Figure 3.15 are shown the dynamic evolution of some one-dimensional cellular automata with different update schemes. The automata have 250 cells and the states of all cells were randomly initialized before each run. All models were evolved for 5000 time steps, where, for the synchronous and cyclic schemes, one time step is completed when each cell has been updated once and for the other schemes, one time step is completed when there have been 250 cell updates.

3.1.6 Dissipative Cellular Automata

The Dissipative Cellular Automata (DCA) are a class of cellular automata that have been defined as dissipative, i.e., cellular automata that are open and makes it possible for the environment to influence their evolution (Zambonelli et al., 2002).

The two main characteristics of the DCA are the asynchronous time-driven dynamics and openness. DCA are Asynchronous Cellular Automata: according to the asynchronous dynamics (Buvel and Ingerson, 1984; Lumer and Nicolis, 1994), at each

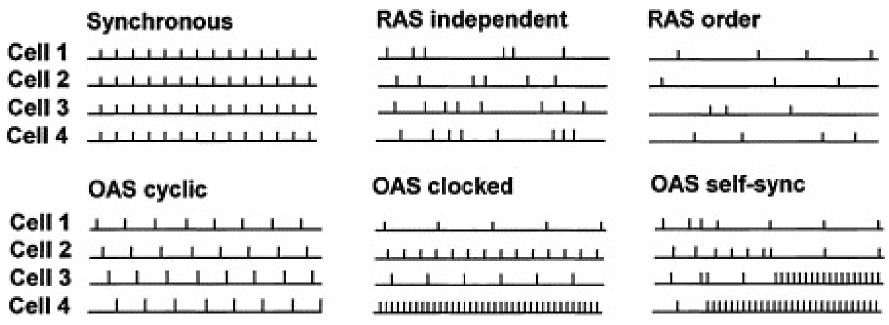


Figure 3.14: *Diagrams of the six asynchronous updating schemes. The horizontal axis shows time, and marks indicate when the cell is updated. (Source: Cornforth et al., 2005, fig. 1, p. 72)*

time, one cell has a probability of rate λ_a to autonomously wake up and update its state.

The above characteristics of modern software systems are reflected in DCA. can be considered as a minimalist open agent system (or, more generally, as a minimalist open software system). As that, the dynamic behavior of DCA is likely to provide useful insight into the behavior of real-world open agent systems and, more generally, of open distributed software systems.

From a more formal point of view, the elements characterizing a dissipative cellular automaton are

- a cellular automata $\mathcal{A} = (\mathcal{L}, \mathcal{S}, \mathcal{N}, f)$,
- an asynchronous dynamics (with uniform distribution of rate λ_a);
- a perturbation action $\varphi(\alpha, \mathcal{D}, \lambda_e)$.

The updating of a cell is an atomic operation and is mutually exclusive among neighbors, without preventing non-neighbor cells to update their state concurrently. The dynamic behavior of the DCA can be influenced by the external environment: some cells can be forced from the external to change their state. The perturbation action φ is a transition function that can change the state of any cell to a given state α according to a probabilistic distribution $\mathcal{D}$. The state change imposed by the perturbation action is independent from the current state of the of the cell and its neighbors.

3.1.7 Cellular Automata With Memory

Standard CA are ahistoric (memoryless): the cells have no memory of previous states, except the last one in the case the central cell is included in the neighborhood. Historic memory can be embedded in CA increasing the number of states and modifying the transaction function. Alonso-Sanz proposed to maintain the transaction rule unaltered, but make them act not only to the current state but weighted mean value of

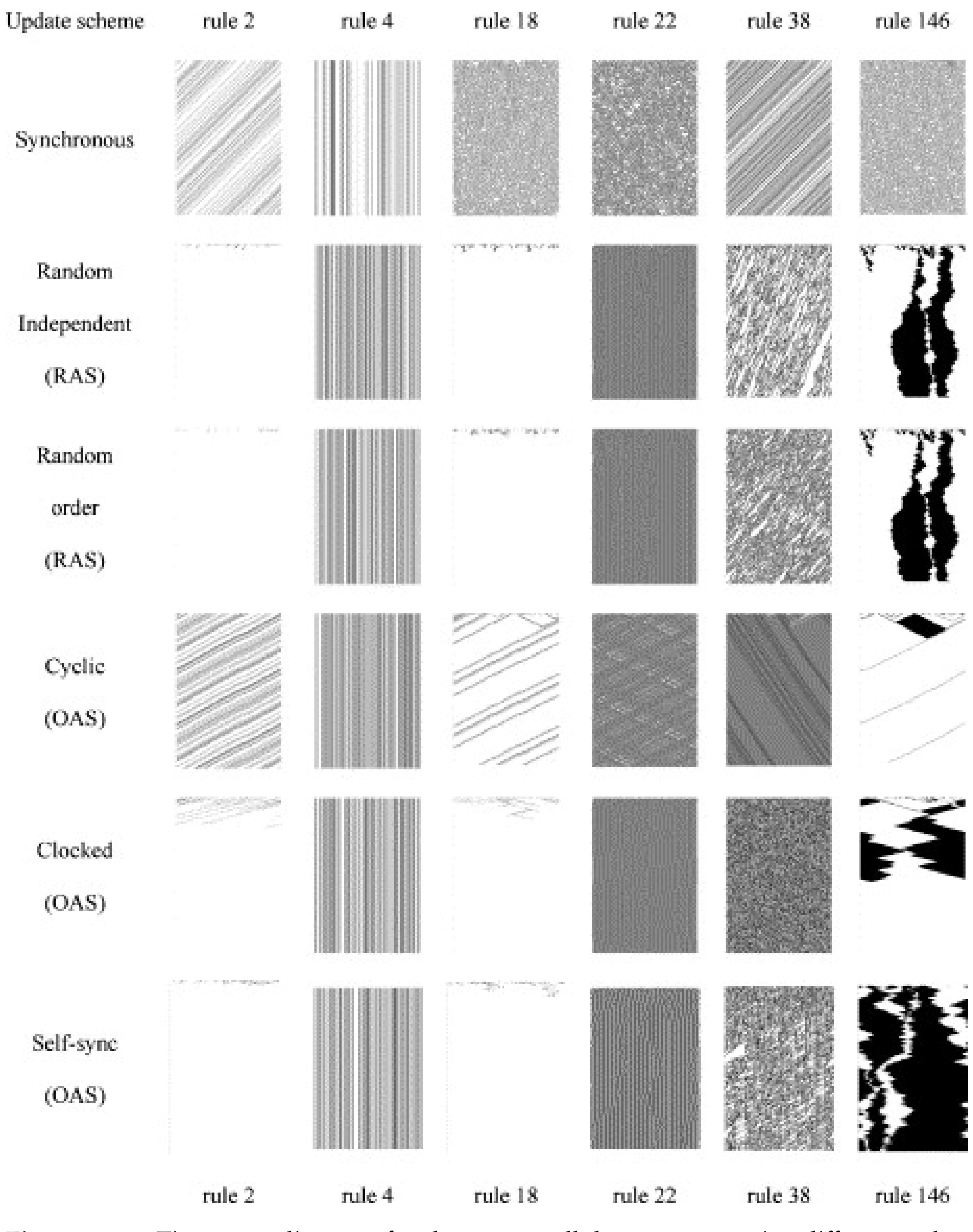


Figure 3.15: *Time space diagrams for elementary cellular automata using different update schemes. The rules shown represent different classes of behavior. Rules 2 and 38: cyclic attractor, rule 4: point attractor, rules 18, 22 and 146: chaotic behavior. (Source: Cornforth et al., 2005, fig. 2, p. 76)*

their previous states (Alonso-Sanz and Martín, 2002; Alonso-Sanz and Martin, 2003; Alonso-Sanz, 2003, 2004, 2006, 2007; Alonso-Sanz and Cardenas, 2008). The values mean for the cell, m_c is calculated as:

$$m_c^{(t)} = \frac{\sum\limits_{i=1}^{t} \alpha^{t-i} \sigma_c^{(t)}}{\sum\limits_{i=1}^{t} \alpha^{t-i}} \equiv \frac{\omega_c^{(t)}}{\Omega(t)} \tag{3.10}$$

where α is the memory factor. The particular case of $\alpha = 1$ is called *full memory*. For the binary cellular automata, geometrically discounted memory does not have an effect if $\alpha \leq 0.5$. If $\alpha \geq 0.61805$, a cell with state history 001 or 110 will be featured after $t = 3$ as 0 and 1 respectively instead of 1 and 0. The patterns of the ahistoric and historic models typically diverge where $t > 3$.

The memory mechanism proposed by Alonso-Sanz is accumulative in its demand of knowledge of past history: to calculate the memory charge $\omega_c^{(t)}$ it is not necessary to know the whole $\{\sigma_c^{(t)}\}$ series, while it suffices to sequentially proceed as: $\omega_c^{(t)} = \alpha\omega_c^{(t-1)} + \sigma_c^{(t)}$.

In the two-state scenario, the s values are obtained as:

$$s_c^{(t)} = \begin{cases} \sigma_c^{(t)} & \text{if} \quad m_i^{(t)} = \frac{1}{2} \\ \text{round}(m_i^{(t)}) & \text{otherwise} \end{cases} \tag{3.11}$$

Figure 3.16 shown an example of the effect of different memory factors an the same rule (the parity rule) in a two-dimensional lattice starting from a single "live" cell. According to the author, CA with memory can be considered as a promising extension of the basic CA paradigm. In some contexts, a transaction rule with memory could math the correct behavior of the CA system of a given complex system.

3.2 Automata Networks

Automata Networks are a generalization of the Cellular Automata, introduced by Ulam, McColloch and von Neumann to study physical and biological phenomena (Mcculloch and Pitts, 1943; Ulam, 1962; von Neumann, 1966). Informally, the automata network are similar to a cellular automata in which the space is a graph instead of a regular lattice. The research in the field was oriented towards formalizing the complexity of automata measured by their computing capabilities (Golès and Martinez, 1990). In many cases, the dynamic evolution of an automaton network is impredictable and the only way to know its evolution is to simulate the automaton.

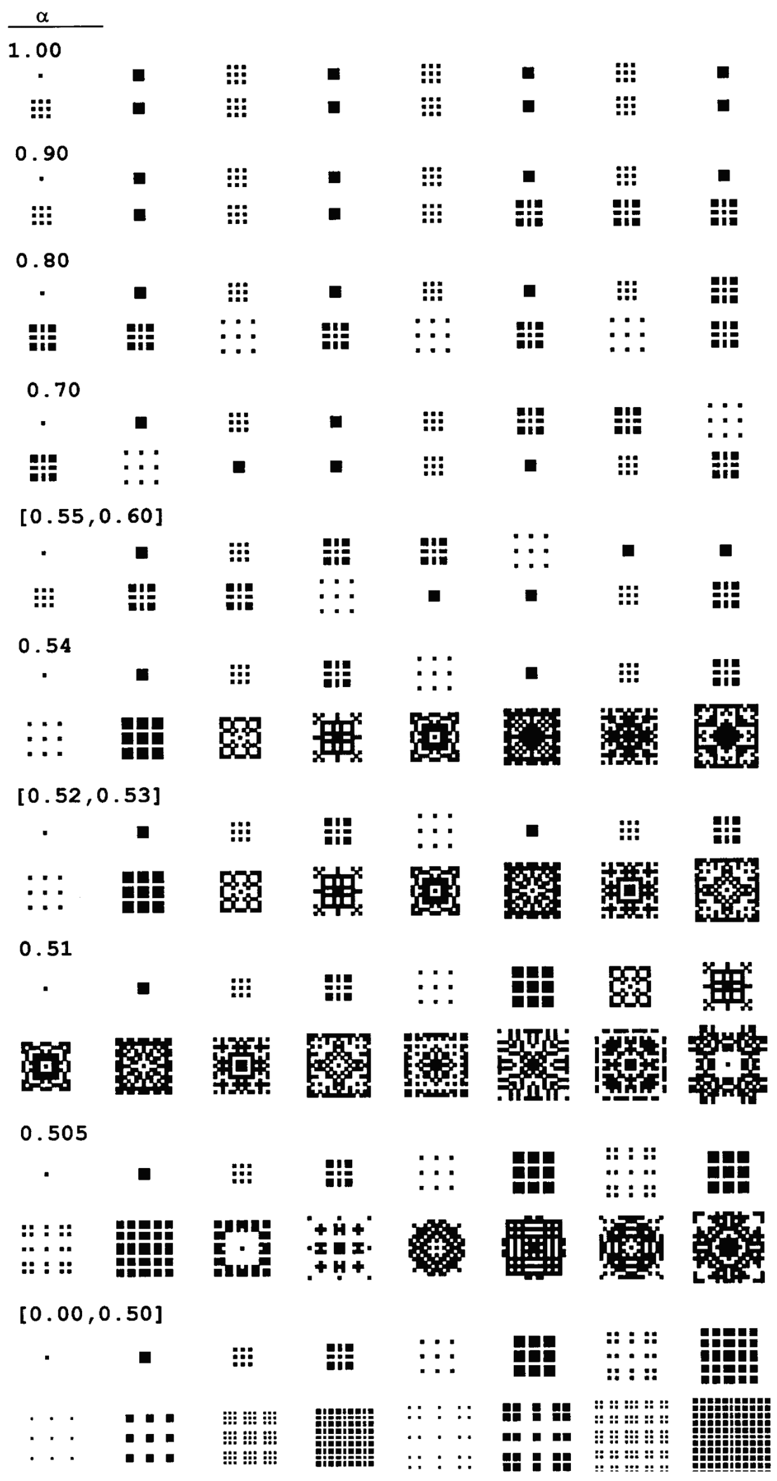


Figure 3.16: *The effect of different memory factors (α) starting with a single site live cell when the parity rule is applied in a two-dimensional lattice, Moore Neighborhood with radius 1. (Source: Alonso-Sanz and Martín, 2002, fig. 4, p. 225)*

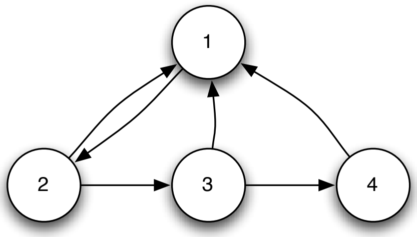


Figure 3.17: *Example of the XOR network graph.*

3.2.1 Formal Definition

We call automata network the triple $(\mathcal{G}, \mathcal{S}, (f_i : i \in \mathcal{I}))$ where

- $\mathcal{G} = (\mathcal{I}, \mathcal{V})$ is a graph on the set of sites $\mathcal{I}$ with connection given by the set of arcs $\mathcal{V} \subset \mathcal{I} \times \mathcal{I}$,
- $\mathcal{S}$ is the set of states, which in most cases is assumed to be finite,
- $f_i : \mathcal{Q}^{|v_i|} \to \mathcal{Q}$ is the transaction function associated to the vertex i.

The graph $\mathcal{G}$ is assumed to be locally finite, which means that $\mathcal{V}_i = \{j \in \mathcal{I} : (j, i) \in \mathcal{V}\}$ is finite. The update rule of an automaton can take different form, for example, it could be parallel or sequential. These two kind of updating may have completely different dynamic behaviors. A detailed analysis on the different kinds of iterations are presented in Robert (1986).

In a parallel iteration, all the sites of the graph $\mathcal{G}$ are update at the same time, so the dynamics of the network $\mathcal{A}$ is given by the equation

$$x_i(t+1) = f_{\mathcal{A}i}(y_i : j \in \mathcal{V}_i) \tag{3.12}$$

In a sequential iteration the sites of the networks are updated one at a time, in a predefined order, given by the relation $\leq$ on $\mathcal{I}$, so the dynamics of the network $\mathcal{A}$ is given by the equation

$$x_i(t+1) = f_{\mathcal{A}i}(x_j(t) : j \in \mathcal{V}_i) \tag{3.13}$$

where

$$y_i = \begin{cases} x_j(t+1) & \text{if} \quad j < i \\ x_j(t) & \text{if} \quad j \geq i \end{cases} \tag{3.14}$$

The different update strategies influenced the dynamic evolution of the networks. In order to better explain this concept, we introduce an example of automata network, called XOR Network. Let $\mathcal{A} = (\mathcal{G}, \mathcal{S}, f_i)$ be a finite automaton where

- $\mathcal{G}$ is the graph shown in Figure 3.17;

- $\mathcal{S} = \{0, 1\}$;
- $f_1(x_2, x_3, x_4) = (x_2 + x_3 + x_4) \bmod 2$
- $f_2(x_1) = x_1$;
- $f_3(x_2) = x_2$;
- $f_4(x_3) = x_3$.

Having 4 nodes with 2 possible states, there are $2^4 = 16$ possible states for the network. In the iteration graphs, each state is identified with $x \in \mathbb{N}$, where

$$x = \sum_{i=1}^{4} x_i 2^{(i-1)} \tag{3.15}$$

The iteration graph for the synchronous update is show in Figure 3.18, in Figure 3.19 is shown the update order $1 < 2 < 3 < 4$ and, in Figure 3.20 the update order $4 < 3 < 2 < 1$.

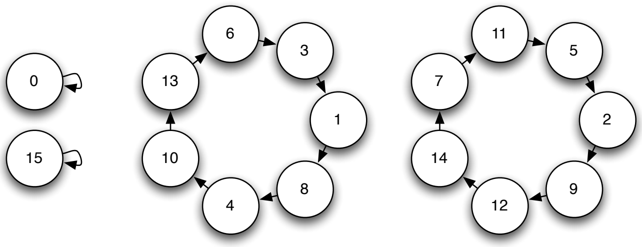


Figure 3.18: *Iteration graph of the synchronous XOR network.*

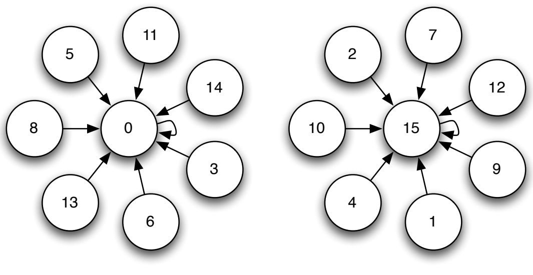


Figure 3.19: *Iteration graph of the XOR network with sequential update order* $1 < 2 < 3 < 4$.

3.2.2 Multilayered Automata Networks

Multilayered Automata Networks (MAN) have been defined in Bandini and Mauri (1999) as a generalization of Automata Networks. MAN were introduced for modeling complex biological process, for example simulating the dynamic regulation of

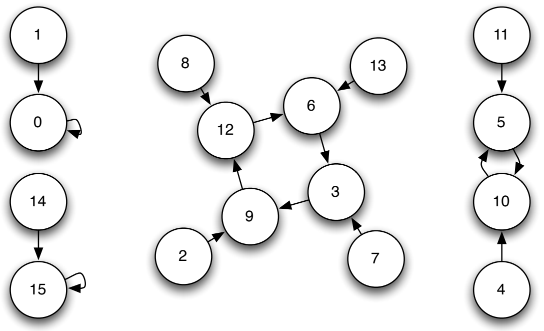


Figure 3.20: *Iteration graph of the XOR network with sequential update order* $4 < 3 < 2 < 1$.

Ca^{2+} distribution in the subcompartments of a living cell. The main features of the Multilayered Automata Network are the explicit introduction of a hierarchical structure based on nested graphs. Such graphs are composed of vertices and edges where each vertex can be in turn be a nested graph of lower level. If we have a graph of level 0, then each node will be a finite state automaton, while for a graph of level $k > 0$, every node will correspond to an automata network of level $k - 1$. In the particular case in which $k = 0$ the multilayered automata network has only one level, giving an automata network in the sense of (Golès and Martinez, 1990).

A Multilayered Automata Network of level 0 ($\mathcal{A}^0$) is a triple $(\mathcal{G}, \mathcal{S}, \mathcal{F})$ where

- $\mathcal{G}^0 = (\mathcal{V}^0, l^0) \in \mathscr{G}$ is the support graph of A^0;
- $\mathcal{S}^0$ is the finite set of states;
- $\mathcal{F}^0 = \{f_v^0 | v \in \mathcal{V}^0\}$ is a set of functions, where $f_v^0 : (\mathcal{S}^0)^{l_v} \to \mathcal{S}^0$.

A MAN of level i ($\mathcal{A}^i$) is a fourtuple $(\mathscr{HG}^i, \mathcal{S}^i, \mathcal{F}^i, g^i)$ where

- $\mathscr{HG}^i = (\mathcal{G}^i, \varphi^i) \in \mathscr{HG}_i, \mathcal{G}^i = (\mathcal{V}^i, l^i) \in \mathcal{G}$ is the support graph of A^i;
- $\mathcal{S}^i$ is a finite set of states the cells of $\mathcal{V}^i$ can assume;
- $\mathcal{F}^i = \{f_v^i | v \in \mathcal{V}^i\}$ is s a set of transaction functions, where $f_v^i : (\mathcal{S}^i)^{l_v^i} \times \mathcal{S}^i \to \mathcal{S}^i$.
- $g^i = \{g_v^i | v \in \mathcal{V}^i\}$ is a set of functions such that $g_v^i : (\mathcal{S}_v^{(i-1)})^{\delta(v)} \to \mathcal{S}^i$ where S_v^{i-1} is the set of states of the network having $\varphi^i(v)$ as its support graph.

Spatial structure Multilayered Automata Network are defined on a hierarchical structure based on nested graphs. Such graphs are composed of vertices and edges where each vertex can be in turn be a nested graph of lower level.

A graph $\mathcal{G}$ is a pair (V, l) where

- V is a finite or countable set of elements called *nodes*;

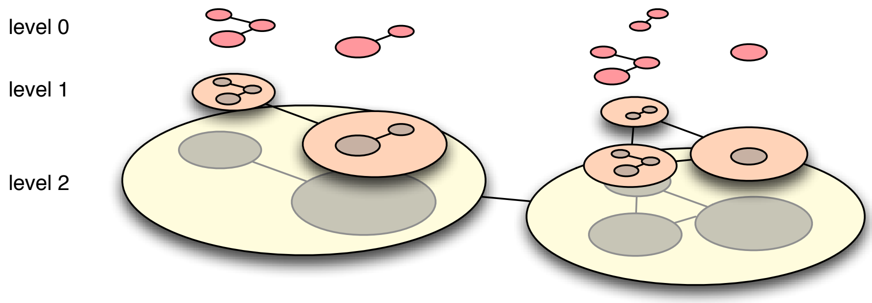


Figure 3.21: *Example of a nested graph of level 2.*

- $l : V \to p(V)$ is the neighborhood function, which determines for each node $v \in V$ the adjacent nodes. For sake of simplicity, the notation l_v is use instead of $l(v)$.

A *graph G* is *locally finite* if and only if $\forall v \in V(|l_v| < \infty)$. The set of graphs will be denoted by $\mathcal{G}$.

The hierarchical structure is recursively defined as a set of *nested graphs*, where the set of nested graph of level 0 is the set of graphs $\mathcal{HG}_0 = \mathcal{G}$ and the set of nested graphs of level $i + 1$ is the set of pairs

$$\mathscr{HG}_{i+1} = \{(\mathcal{G}, \varphi) | \mathcal{G} = (V, l) \in \mathcal{G}, \varphi : V \to \mathscr{HG}_i\} \tag{3.16}$$

where $\mathcal{G}$ is a graph called *supportgraph* of the pair $(\mathcal{G}, \varphi)$ and φ is a map which associates to every node of $\mathcal{G}$ a nested graph of level i. The set $\mathscr{HG}$ of nested graphs is the union of all the sets $\mathscr{HG}_i$. A nested graph of level i will be denoted as $\mathscr{HG}^i$ and each node of the graph will be identified by a pair (i, v) where i denotes the level and v is the identifier of the node in the graph of level i.

The *horizontal neighborhood* of the node (i, v) is defined by the neighborhood function l^i related to the graph of level i.

The *nesting neighborhood* function δ associates to the node (i, v) the set $\delta(v) = V_v^{i-1}$ of nodes of the graph at level $i - 1$:

$$\varphi^i(v) = \begin{cases} \mathscr{HG}_v^{i-1} = (V_v^{i-1}, l_v^{i-1}) & \text{if} \quad i - 1 = 0 \\ \mathscr{HG}_v^{i-1} = ((V_v^{i-1}, li - 1_v), \varphi_v^{i-1}) & \text{if} \quad \text{otherwise} \end{cases} \tag{3.17}$$

As an example, let us consider the nested graph of level 2 shown in Figure 3.21. Both the level 0 and the level 1 are constituted by a family of graphs, each one corresponding to a node at the preceding level, and the level 2 is composed by one connected graph consisting of two nodes.

If we take two nodes of two distinct subgraphs of level 1, there is no path from one to the other given by edges of level 1, but if the two subgraphs correspond to adjacent nodes of level 2, then a path which connects them exists.

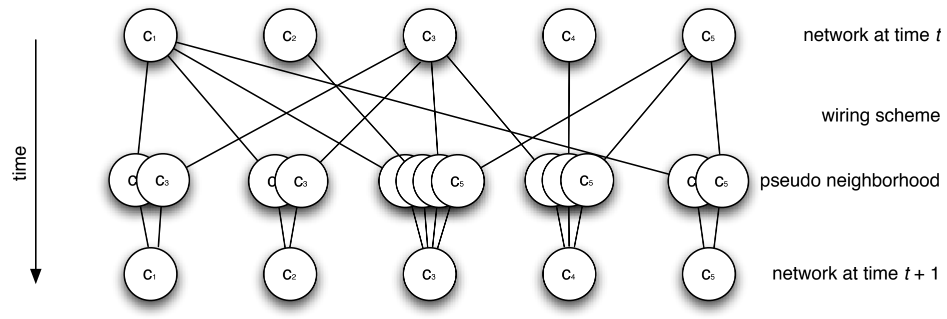


Figure 3.22: *Example of a Random Boolean Network. Each cell updates its value according to the values in the pseudo neighborhood. Each network element may have a different wiring scheme and rule.*

Dynamics Since the "next state" of each node in the network depends on the "current state" of the node and of its (horizontal or vertical) neighbours, the evolution of the networks depends also from the updating policy we choose.

As for the automata networks, it is possible to give different types of dynamics also for the multilayered automata networks, depending on the fact that the state transition occurs in parallel in all the nodes of the network or that only a node is updated at every time step, in a sequential way. In *synchronous iteration* all the nodes are updated in parallel. In the *sequential iteration* the states of the nodes are updated one at a time, by levels from k to 0 and, at every level, following a given schedule.

3.3 Random Boolean Network

Random Boolean Network (RBN) are the most general instance of a synchronous discrete dynamical system. RBN may be viewed as a generalization of CA (Wuensche, 1993), allowing arbitrary wiring scheme and subjective rules for each network cell. An example of a RBN is shown in Figure 3.22.

RBN have been used in diverse areas to model complex systems. RBN were proposed in Kauffman (1969), as models of genetic regulatory networks. The author argues that proto-organisms probably were randomly aggregated nets of chemical reactions. The hypothesis that contemporary organisms are also randomly constructed molecular automata is examined by modeling the gene as a boolean device and studying the behavior of large, randomly constructed nets of these binary genes. The Kauffman results suggest that if in a RBN each gene is directly affected by two or three other genes, the network behaves with great order and stability.

3.3.1 Formal Definition

Formally a RBN is the n-tuple $(\mathcal{L}, \mathcal{S}, \mathcal{N}, f)$ where

- $\mathcal{L}$ is a set of n cells,
- $\mathcal{S} = \{0, 1\}$ is the set of v states (v is equal to 2 for the Boolean Network),
- $\mathcal{N}$ is a finite set of neighbors,
- $\mathcal{F}$ is a set of n transaction function $f_i : \mathcal{S}^n \to \mathcal{S}$.

The global state of a network of n cells is the pattern resulting from values assigned to each element from a finite set of v values. At each time step t, the value of a cell c_i depends on its transaction boolean function f_i, applied to its pseudo neighborhood. The neighborhood of each cell in RBN is determined by its wiring scheme. The number of neighbors k is the same for each node of the network in the case of homogenous connectivity k or different in the case of the mixed k networks. Since duplicated connections are allowed, there are n^k possible wiring scheme. The networking scheme is usually fixed over time. A fully connected RBN is equivalent to a random mapping (Kauffman, 1969).

The time evolution of the cell c_i is given by

$$c_i^{(t+1)} = f_i i(c^t_{w_{i,1}}, c^t_{w_{i,2}}, c^t_{w_{i,3}}, \ldots, c^t_{w_{i,k}}) \tag{3.18}$$

where $c_{w_{i,1}}, \ldots, c_{w_{i,k}}$ are k neighbors of the cell c_i.

RBN have a vastly parameter space, and thus behavior space than CA. According to (Wuensche, 1993), the number of all possible combinations of wiring and rule of a RBN of size n with an homogenous connectivity k is given by

$$S_{n,k} = (n^k)^n \times (2^{2^k})^n \tag{3.19}$$

In "Classic" Random Boolean Networks (CRBN), the updating scheme is synchronous. All the nodes in the network are updated at time $t + 1$ depending on their states at time t.

The dynamics arising from such networks are very interesting, since depending on the values of n and k, they can be ordered, complex, or chaotic (Kauffman, 1993). Diverse properties of CRBNs have been studied, among others, Wuensche (1998) and Aldana et al. (2003).

Since Classic Random Boolean Networks are deterministic, once the state of a network reaches an attractor, it will never have states different from the ones in the attractor.

3.3.2 Classification of Random Boolean Networks

A classification of the RBN is presented in Gershenson et al. (2003). In Figure 3.23 the classification is graphic representation. In order to have a complete taxonomy of RBN, the authors defined several types of RBN, setting all of them under the label Discrete

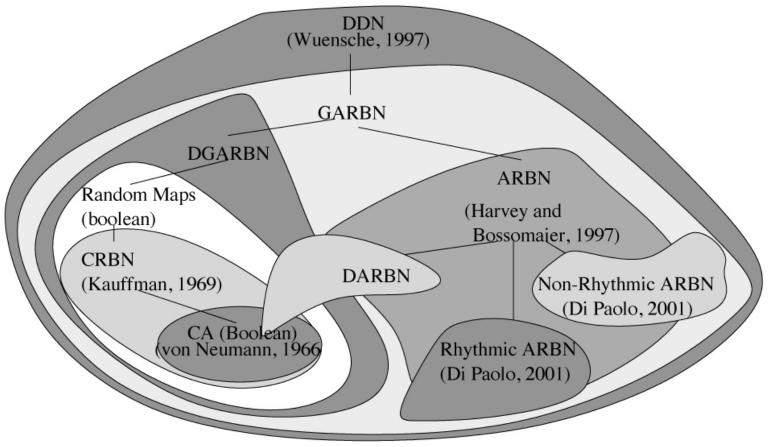


Figure 3.23: *Classification of Random Boolean Networks. (Source: Gershenson et al., 2003, fig. 1, p. 2)*

Dynamic Networks (DDN). DDN have all the properties common in all the types of RBN. In fact, DDN are discrete in terms of time, space, and values.

The most general networks are the Generalized Asynchronous Random Boolean Networks (GARBN), that are as Asynchronous Random Boolean Networks which can update any number of nodes at each time step. GARBN can go from updating only one random node at each time step, to updating all the nodes synchronously.

Asynchronous Random Boolean Networks (ARBN) updating is asynchronous and random. At each time step, a single node is selected at random in order to be updated ARBN are non-deterministic, so there are no cycle attractors, only point and, loose attractors (Harvey and Bossomaier, 1997) which are parts of the state space which also capture the dynamics, but since the updating order of the nodes is random, the order of the states will not be repeated deterministically.

A small subset of all possible ARBN have Rhythmic (Rhythmic ARBN) and Non-Rhythmic (Non-Rhythmic ARBN) attractors. The measure of rhythmic behavior will be a measure of how patterns occurring at different instants in the history of a system relate to one another. For the case of ARBN in particular it is possible to devise a variety of simple measures based on correlations between states occurring at different points during the evolution (Paolo, 2001).

Deterministic Asynchronous Random Boolean Networks (DARBN) (Gershenson et al., 2003) as ARBN which do not select at random which node to update. Each node has associated two parameters: p and q, where $(p, q \in \mathcal{N}, q < p)$. The parameter p determines the period of the update (i.e. how many time steps the node will wait in order to be updated), and the parameter q the translation of the update. A node will be updated when the modulus of time t over p is equal to q. If two or more nodes will be updated at a specific time, the first node is updated, and then the second is updated

net(t)	net($t+1$)
11	11
10	01
01	00
00	10

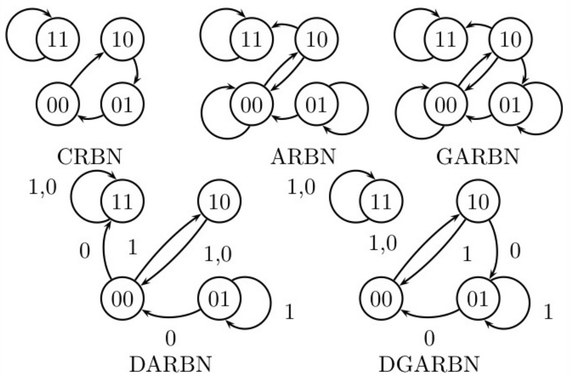


Figure 3.24: *On the left, the transition table for a RBN $n = 2$, $k = 2$. On the right graphs of RBN with different updating schemes. The arrows with numbers indicate which transition will take place depending on the modulus of time over two. (Source: Gershenson et al., 2003, fig. 2, p. 3)*

taking into account the new state of the network.

Also in Deterministic Generalized Asynchronous Random Boolean Network (DGARBN), each node is associated with the two parameters p and q. If more then one node are determined to be updated at the same time step, they will be updated synchronously (i.e. they will all be updated at time $t + 1$ taking into account the state of the nodes at time t).

Boolean Random Maps are obtained when $n = k$, and, for all the nodes of the network, $p = 1$ and $q = 0$.

Classic Random Boolean Network are a subset of random maps and Boolean Cellular Automata are specific cases of CRBN, where the connectivity is limited by the spatial organization of the nodes.

As an example of the different updating schemes, in Figure 3.24 is shown the transition table of a RBN with $n = 2$, $k = 2$, and the network evolution according to the different schemes.

The only reason for time is so that everything doesn't happen at once.

Albert Einstein

4 Effects of Asynchrony on Cellular Automata

In this chapter we present a study of the effect of asynchrony on Cellular Automata. The aim of this study is to introduce the problematics deriving from the adoption of an asynchronous CA model. In the first section, we present several cellular automata update schemes and a tentative classification of such schemes. In order to study the effects of the different update schemes, we introduced a class of simple CA, called One Neighbor Binary Cellular Automata (1nCA). We firstly overview the general features of 1nCA, than we present the effects of 6 different updates schemes on all the class of 1nCA.

4.1 Asynchronous Cellular Automata

Cellular Automata have traditionally treated time as discrete and state updates as occurring synchronously and in parallel. Moreover, several authors (e.g. Paolo, 2000; Thomas and Organization., 1979) have argued that asynchronous models are viable alternatives to synchronous models and suggest that asynchronous models should be preferred where there is no evidence of a global clock.

There are several cellular automata asynchronous update schemes. Kanada (1994) introduce an asynchronous model, which is called 1D-ACA. This CA update scheme is basically a sequential model, in which one cell is update in each time step. The order of updating sequence is defined by one of the following three methods:

- *Random order* The elements of the updating sequence are random.
- *Fixed Random Order* The first N, where N in the number of cells, elements of the sequence are random numbers, and these values are repeated in the sequence. Thus, the sequence is periodic
- *Interlaced order*: The index of the cell to be update at time t is calculated as $C\,t\,mod\,N$, where C is a parameter and prime to N.

Cornforth et al. (2005) identify three classes of update scheme: Synchronous Update Random Asynchronous (RAS), and Ordered Asynchronous (OAS). According to the Random Asynchronous, at any given time individuals to be updated are selected at random according to some probability distribution. In the Ordered Asynchronous update process, the updating of individual states follows a systematic pattern. The authors consider a total of six update patterns, including two RAS schemes and three OAS scheme.

- *Synchronous Scheme* All individuals are updated in parallel at each time step.
- *Random Independent (RAS)* At each time step, the cell to be updated is chosen at random.
- *Random Order (RAS)* All nodes are updated in random order. After the updating off all the nodes, the order is changed.
- *Cyclic (OAS)* At each time step a node is chosen according to a fixed update order, which was decided at random.
- *Clocked (OAS)* A timer is assigned to each cell, so that updating is autonomous and proceeds at different rates for different cells.
- *Self-Sync* This model is similar to the clocked scheme, but incorporates local synchrony. The author chose to implement local synchrony by using a coupled oscillator approach. The period of each timer is adjusted after an update so as to more closely match the period of other cells in its neighbourhood.

Starting from the previously presented works, we create a classification of the update schemes. In order to classify the update schemes, we define the following parameters:

- $p_i^{(t)}$ determines the period of the update of the cell i at the time step t, i.e. how many time steps the cell i will wait in order to be updated. The value of p can change during the time, e.g. in the Self-Sync update scheme.
- $l_i^{(t)}$ determines the length (in terms of time step) of the updating of the cell i at the time step t, i.e. after how many time steps the neighbor cells taking into account the new state during their updated.
- d_i, determines the delay (in terms of time step) before the first update.
- $U^{(t)}$ is the set of cells beginning the updating process at the at the time step t.
- $u^{(t)} = |U^{(t)}|$, is the number of cells beginning the updating process at the time step t.

We introduce a simple example in order to explain the meaning of such parameters. Let consider the cellular automaton shown in Figure 4.1. It is a one dimension cellular automaton composed of 4 cells. The neighborhood size is 2 (radius 1), so each cells has almost 2 neighbors. The state of each cell is an integer number between 0 an 100. The update rule is the following: the new cell state is the sum of the previous cell state and the neighbors cell modulo 100.

- $\mathcal{L} = \{c_0, c_1, c_2, c_3\}$ is a one dimensional array of cells,
- $\mathcal{S} = 0 \ldots 100$ is the set states,
- $\mathcal{N} = \{\{c_0, c_1\}, \{c_1, c_0\}, \{c_1, c_2\}, \{c_1, c_2\}, \{c_2, c_3\}, \{c_3, c_2\},\}$ is the set of neighbors,
- $f(s, n_1, n_2) = (a + n_1 + n_2)\ mod\ 100$ is a transition function for the cell with 2 neighbors, for the cell with only one neighbor, the transition function is $f(s, n_1) = (a + n_1)\ mod\ 100$.

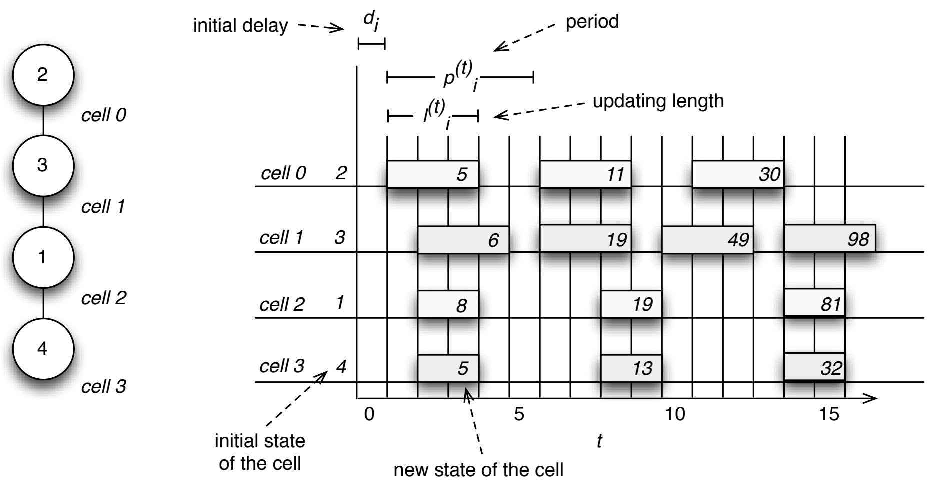


Figure 4.1: *An example of cellular automata composed of 4 cell and its evolution according to the asynchronous clocked update scheme.*

In the following section we present several update schemes. We each update scheme, we give a formal definition. These formal definition are successively employed for the schemes classification.

4.1.1 Synchronous Scheme

All individuals are updated in parallel at each time step, as shown in Figure 4.2. The updating of a cell takes 1 time step. More formally:

$$\forall\, t \in \mathbb{Z}\ \ t > 0\ \ \forall\, i \in \mathbb{Z}\ \ 0 \leq i < N\ \ p_i^{(t)} = 1\ \ l_i^{(t)} = 1\ \ d_i = 0\ \ u^{(t)} = N \tag{4.1}$$

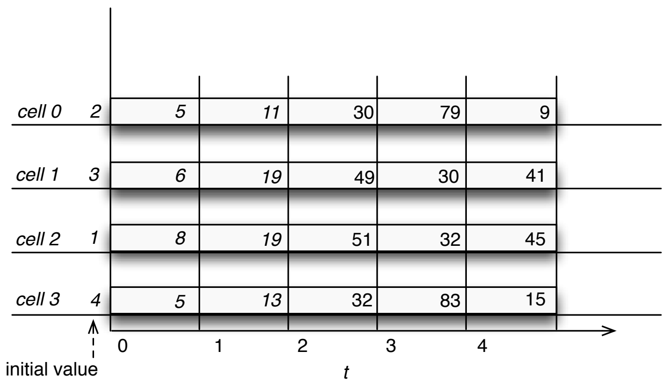


Figure 4.2: *The evolution of the cellular automata according to the synchronous update scheme.*

4.1.2 Random Independent

At each time step, one and only cell, chosen at random, is updated. The updating of a cell takes 1 time step. The evolution of the example automata according to the Random Independent update scheme is shown in Figure 4.3. More formally:

$$\forall\, t \in \mathbb{Z}, t > 0 \quad \forall\, i \in \mathbb{Z} \quad 0 \le i < N \quad l_i^{(t)} = 1 \quad u^{(t)} = 1 \quad \exists\, t, i \quad p_i^{(t)} > 1 \tag{4.2}$$

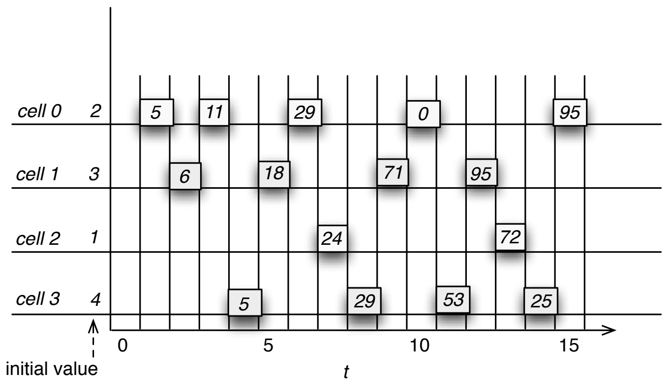


Figure 4.3: *The evolution of the cellular automata according to the Random Independent update scheme.*

4.1.3 Random Order

All nodes are updated in random order. After the updating off all the nodes, the order is changed. The updating of a cell takes 1 time step. The evolution of the example automata according to the Random Independent update scheme is shown in Figure 4.4.

The maximum length of the update period is less than $2N$. More formally:

$$\forall\, t \in \mathbb{Z}\;\; t > 0 \;\;\; \forall\, i \in \mathbb{Z}\;\; 0 \le i < N \;\;\; p_i^{(t)} < 2N \;\;\; l_i^{(t)} = 1 \;\;\; d_i < N \;\;\; u^{(t)} = 1 \tag{4.3}$$

We can define an update interval $[\alpha, \omega]$ as:

$$\forall\, z \in \mathbb{Z}, z > 0 \quad \alpha = 1 + z\, N \quad \omega = (z+1)\, N \tag{4.4}$$

In every update interval, each cell is update exactly one time:

$$\begin{gathered}\forall\, i \in \mathbb{Z} \quad 0 \le i < N \quad \forall\, t_n \in \mathbb{Z}, \alpha \le t_n \le \omega, \quad \forall\, t_m \in \mathbb{Z}, \alpha \le t_m \le \omega, \\ c_i \in U^{(t_n)},\; c_i \in U^{(t_m)} \iff t_n = t_m\end{gathered} \tag{4.5}$$

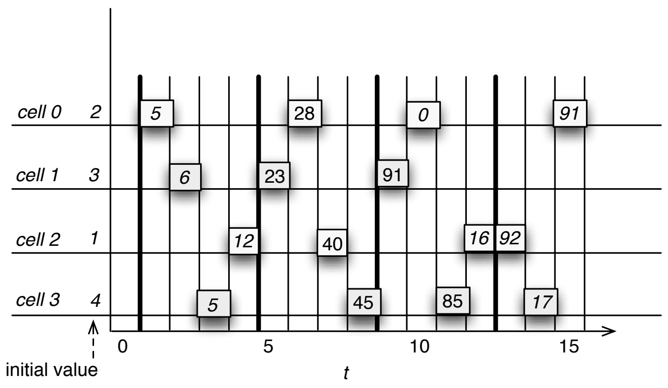


Figure 4.4: *The evolution of the cellular automata according to the RAS Random Order update scheme.*

4.1.4 Cyclic

As shown in Figure 4.5, at each time step a node is chosen according to a fixed update order. Formally:

$$\forall\, t \in \mathbb{Z}\;\; t > 0 \;\;\; \forall\, i \in \mathbb{Z}\;\; 0 \le i < N \;\;\; p_i^{(t)} = N \;\;\; l_i^{(t)} = 1 \;\;\; d_i < N \;\;\; u^{(t)} = 1 \tag{4.6}$$

We can identify two subtypes of this update scheme:

- *Random Cyclic* The update order is decided at random during initialisation of the automaton. This update scheme correspond to the Kannada's *Fixed Random* (Kanada, 1994) and Cornforth's *Cyclic OAS* (Cornforth et al., 2005).

- *Fixed Cyclic* The update order is fixed in the automaton definition. The *Sequential Ordered* and the *Interlaced Cyclic* belong to this update type:

– *Sequential Ordered* The cells are updated one-by-one according to their natural order:

$$d_i = 1 + i; \quad q^t = (t-1)\, mod\, N \quad u^{(t)} = \{c_{q^t}\} \tag{4.7}$$

– *Interlaced Cyclic* Called *Interlaced Order* in Kanada (1994). The set of cell $u^{(t)}$ to be update at time step t is calculated as

$$q^t = C\,(t-1)\, mod\, N \quad u^{(t)} = \{c_{q^t}\} \tag{4.8}$$

where C is a parameter prime to N.

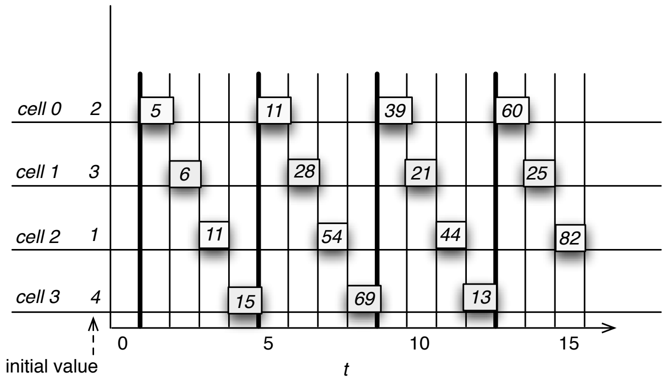


Figure 4.5: *The evolution of the cellular automata according to the Cyclic update scheme.*

4.1.5 Generic Cyclic

Is a generalization of the Cyclic update scheme, obtained relaxing the constraint on the updating length. In this update scheme, the updating length is limited only by the period. Formally:

$$\forall\, t \in \mathbb{Z} \;\; t > 0 \;\; \forall\, i \in \mathbb{Z} \;\; 0 \leq i < N \;\; p_i^{(t)} = N \;\; l_i^{(t)} \leq p^{(t)} \;\; d_i < N \;\; u^{(t)} = 1 \tag{4.9}$$

4.1.6 Clocked

A *timer* is assigned to each cell, so that updating is autonomous and proceeds at different rates for different cells, as shown in Figure 4.6. The update frequency of each cells is fixed:

$$\forall\, t \in \mathbb{Z} \;\; t > 0 \;\; \forall\, i \in \mathbb{Z} \;\; 0 \leq i < N \;\; p_i^{(t)} = p_i^{(0)} \;\; l_i^{(t)} \leq p_i^{(0)} \;\; d_i \leq p_i^{(0)} \tag{4.10}$$

As subtype of the Clocked update scheme is the *Equal Frequency Clocked*. According to this update scheme, every cells has the same update frequency:

$$\forall\, t \in \mathbb{Z} \;\; t > 0 \;\; \forall\, i \in \mathbb{Z} \;\; 0 \leq i < N \;\; p_i^{(t)} = p_0^{(0)} \tag{4.11}$$

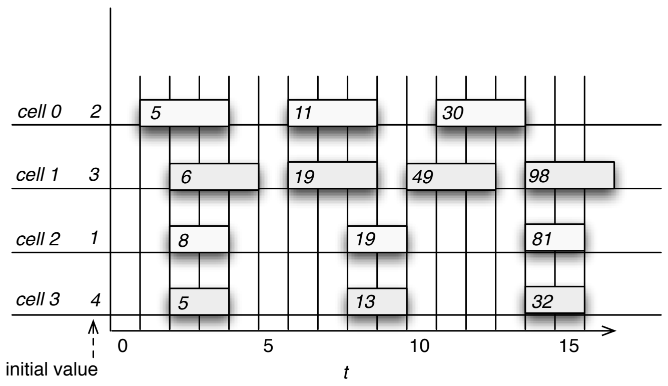


Figure 4.6: *The evolution of the cellular automata according to the Clocked update scheme.*

4.1.7 Generic Clocked

Is a generalization of the Cyclic update scheme, obtained relaxing the constraint on the fixed update frequency. The two subtypes of this update scheme are the *Clocked* and *Variable Clocked*.

According to the Variable Clocked scheme, a timer is assigned to each cell, so that updating is autonomous and proceeds at different rates for different cells. The updating frequency is not fixed:

$$\exists t, i : p_i^{(t)} \neq p_i^{(0)} \tag{4.12}$$

The *Self-Sync* update scheme is an example of Variable Clocked scheme.

4.1.8 CA Update Schemes Ontology

In this section we present a tentative classification of the previously presented update schemes. In order to manage the complexity deriving from the classification of the schemes according to different features (e.g. the number of cells updated at each time step, the maximum length of the update period) we create the CA Update Scheme Ontology. The ontology is expressed using the OWL 2 DL[1] language, a W3C endorsed format that can be adopted to define ontologies. This language allows defining relatively rich semantics including relations between classes of entities, cardinality, equality, properties, characteristics of properties, and enumerated classes.

The CA Update Scheme Ontology consists a set of classes and properties. We defined the following OWL data properties:

- *hasClock* type of *clock* for the timer-based update schemes;
- *hasInitialDelay* length of the initial delay;
- *hasOrder* order of the update sequence;

[1]http://www.w3.org/TR/owl2-primer/

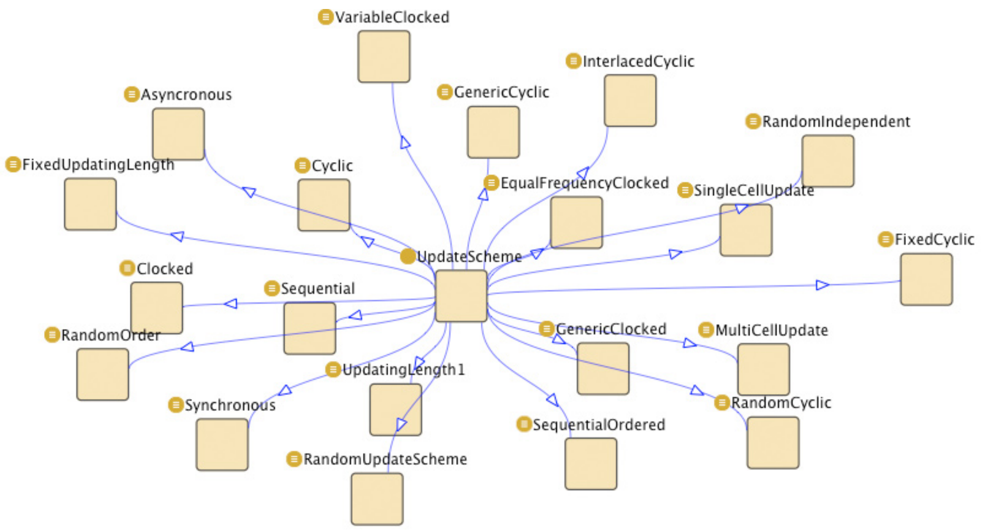


Figure 4.7: *A schematic representation of the classes of the CA Update Scheme Ontology before the classification.*

- *hasPeriod* length of the update period;
- *hasRandomComponent* this boolean properties has the *true* value if and only if the update scheme has some random parameters.
- *hasUpdatingLength* length of the cells updating;
- *updatesCellsPerTimeStep* number of cells updated each time step.

The domain of such properties is the *UpdateScheme* class, the root class of our ontology. We defined the following classes:

Asyncronous $\equiv$ *UpdateScheme* $\sqcap \neg$ *Synchronous*

Asynchronous update scheme, defined as any update scheme that is not synchronous.

Clocked $\equiv$ *UpdateScheme* $\sqcap$
$\neg(\exists\, hasClock.\{\text{variable}\})\sqcap$
$\exists\, hasClock.string\sqcap$
$\exists\, updatesCellsPerTimeStep.\{*\}$

Clocked update scheme. The update frequency of each cells is fixed (not variable).

Cyclic $\equiv$ *UpdateScheme* $\sqcap$
$\exists\, hasPeriod.\{\text{N}\}\sqcap$

$\exists$ *hasUpdatingLength*.{1}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

Cyclic update scheme with period N. Only one cell per time step is updated and updating length is equal to 1

EqualFrequencyClocked $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasClock*.{eq}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{*}

Clocked update scheme. The update frequency is the same for all the cells.

FixedCyclic $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasPeriod*.{N}$\sqcap$
$\exists$ *hasRandomComponent*.{false}$\sqcap$
$\exists$ *hasUpdatingLength*.{1}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

Cyclic, non-random update order with period N. Only one cell per time step is updated and updating length is equal to 1

FixedUpdatingLength $\equiv$ *UpdateScheme* $\sqcap$
$\neg$($\exists$ *hasUpdatingLength*.{*})

Update scheme with fixed cell updating length.

GenericClocked $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasClock.string*$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{*}

Generic clocked update scheme. Any number of cells can start the update at the same time step.

GenericCyclic $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasPeriod*.{N}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

The instances of this class have cyclic update order. Only one cell per time step starts the update at each time step.

InterlacedCyclic $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasOrder*.{interlaced}$\sqcap$
$\exists$ *hasPeriod*.{N}$\sqcap$
$\exists$ *hasRandomComponent*.{false}$\sqcap$
$\exists$ *hasUpdatingLength*.{1}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

Cyclic interlaced update order with period N. Only one cell per time step is updated and updating length is equal to 1

MultiCellUpdate $\equiv$ *UpdateScheme* $\sqcap$
$\neg$ *SingleCellUpdate*

The instances of this class are update scheme which update more than one cell for each time step.

RandomCyclic $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasPeriod*.{N}$\sqcap$
$\exists$ *hasRandomComponent*.{true}$\sqcap$
$\exists$ *hasUpdatingLength*.{1}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

Cyclic random update order with period N. Only one cell per time step is updated and updating length is equal to 1

RandomIndependent $\equiv$ *UpdateScheme* $\sqcap$
$\neg$ *RandomOrder* $\sqcap$
$\neg$($\exists$ *hasPeriod*.{N})$\sqcap$
$\exists$ *hasRandomComponent*.{true}$\sqcap$
$\exists$ *hasUpdatingLength*.{1}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

Random independent update order.
Only one cell per time step is updated and updating length is equal to 1

RandomOrder $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasPeriod*.{¡2N}$\sqcap$
$\exists$ *hasRandomComponent*.{true}$\sqcap$
$\exists$ *hasUpdatingLength*.{1}$\sqcap$
$\exists$ *updatesCellsPerTimeStep*.{1}

Random order with period $< 2N$.
Only one cell per time step is updated and updating length is equal to 1

RandomUpdateScheme $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasRandomComponent*.$\{\mathsf{true}\}$

The instances of this class are update schemes with a random component.

Sequential $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasUpdatingLength*.$\{1\}$
$\exists$ *updatesCellsPerTimeStep*.$\{1\}$

Sequential updating: only one cell per time step is updated and updating length is equal to 1

SequentialOrdered $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *hasOrder*.$\{\mathsf{sequential}\}\sqcap$
$\exists$ *hasPeriod*.$\{\mathsf{N}\}\sqcap$
$\exists$ *hasRandomComponent*.$\{\mathsf{false}\}\sqcap$
$\exists$ *hasUpdatingLength*.$\{1\}\sqcap$
$\exists$ *updatesCellsPerTimeStep*.$\{1\}$

Cyclic, sequential update order.

SingleCellUpdate $\equiv$ *UpdateScheme* $\sqcap$
$\exists$ *updatesCellsPerTimeStep*.$\{1\}$

The instances of this class are update scheme which update only one cell for each time step.

Synchronous $\equiv$ *UpdateScheme* $\sqcap$
$\neg(\exists$ *hasClock.string*$)\sqcap$
$\exists$ *hasInitialDelay*.$\{0\}\sqcap$
$\exists$ *hasPeriod*.$\{1\}\sqcap$
$\exists$ *hasRandomComponent*.$\{\mathsf{false}\}\sqcap$
$\exists$ *hasUpdatingLength*.$\{1\}\sqcap$
$\exists$ *updatesCellsPerTimeStep*.$\{\mathsf{N}\}$

Synchronous update scheme.

UpdateScheme $\sqsubseteq \top$

This is the root class of the CA Update Schemes.

UpdatingLength1 $\equiv \exists\, hasUpdatingLength.\{1\}$

Update scheme with cell updating length equals to 1.

VariableClocked $\equiv$ *UpdateScheme* $\sqcap$
$\exists\, hasClock.\{\text{variable}\} \sqcap$
$\exists\, updatesCellsPerTimeStep.\{*\}$

Clocked update scheme. The update frequency of each cells is variable.

A schematic representation of the ontology is shown in Figure 4.7.

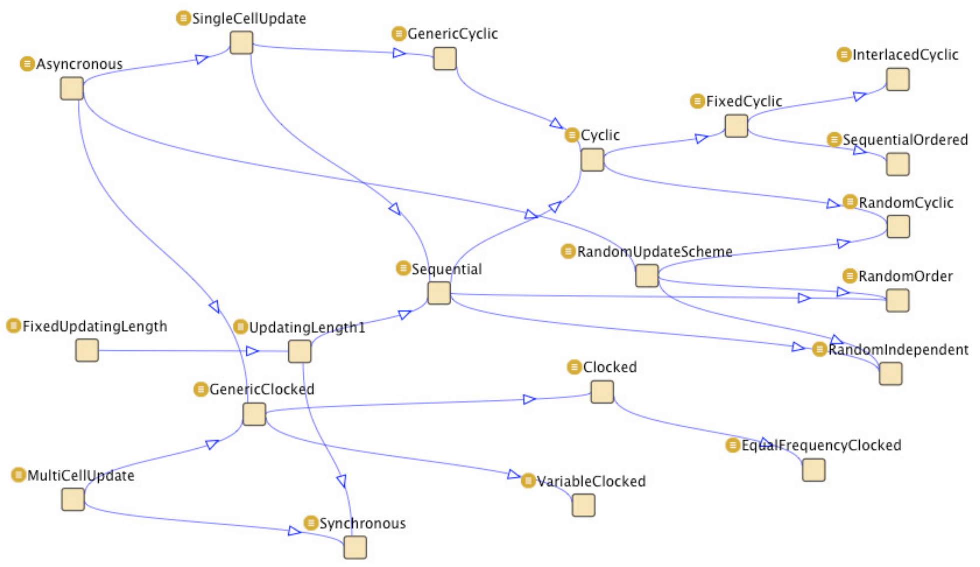


Figure 4.8: *A schematic representation of the classes of the CA Update Scheme ontology after the automatic classification.*

We computed the class hierarchy using the Pellet[1] Semantic Reasoner, an open source Java reasoner for OWL 2 DL. A reasoner is a software able to infer logical consequences from a set of asserted axioms. An example of inference is the compute of the class hierarchy. Starting from the update scheme defined as OWL classes, a reasoner is able to inter how update scheme are subtypes of other schemes. The result of

[1]http://clarkparsia.com/pellet

such classification is shown in Figure 4.8. As shown in the figure, several classes are subclasses of more than one class.

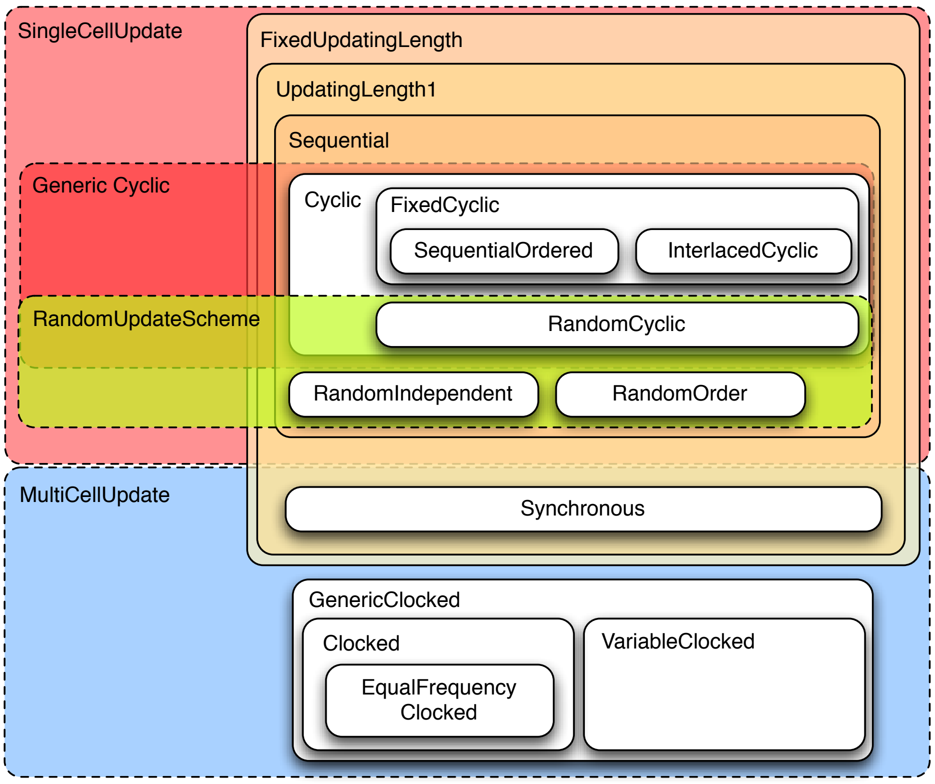


Figure 4.9: *A schematic representation of the classes of the CA Update Scheme ontology. The classes* UpdateScheme *and* Asynchronous *are not shown for simplicity.*

We reorganized the classes in the graph shown in Figure 4.9. An update schema can update one cell per time step (*SingleCellUpdate*) or more than one (*MultiCellUpdate*). The clocked update schemes (*GenericClocked*) and the synchronous update scheme update more than one cell per time step. The schemes that update the cells in sequence (*Sequential*) are subclass of the schemes in which the update of the cell takes exactly one time step (*UpdatingLength1*). This class is a subclass of the update scheme in which the update of the cells takes always a fixed amount of time step (*FixedUpdatingLength*). The synchronous update scheme is a sequential update scheme. The cyclic schemes (*Cyclic*) are both *Sequential* and *GenericCyclic*. The two subclasses of *Cyclic* are *FixedCyclic* and *RandomCyclic*. The *RandomCyclic* class is also a subclass of *RandomUpdateScheme*, The *RandomUpdateScheme* instances are update schemes with a random component, such *RandomIndependent* and *RandomOrder*. This two classes are also instance of sequential because they update only one cell per time step.

4.2 One Neighbor Binary Cellular Automata

One Neighbor Binary Cellular Automata (1nCA) is a one-dimensional Cellular Automata, with two possible states per cell. Each cell has two neighbors, left and right, defined to be the adjacent cells on either side, but the update rule consider only one neighbor per step. The neighborhood includes the cell itself and the left or the right adjacent cell and alternates between these two situations at even and odd time steps.

The size of the neighborhood is always 2, so there are 4 possible patterns for the neighborhood and only 16 possible rules. The number of possible rules is small compared to the 256 possible rules of the Elementary Cellular Automata, so it is easier to exhaustively study the dynamic behavior of the all rules.

These 16 1nCA rules will be referred using the Wolfram notation, with the rule numbers followed by the $\triangle$ symbol to avoid confusion with the Elementary Cellular Automata rules (e.g. "Rule 10" is an Elementary Cellular Automata rule, "Rule $10\triangle$" is an 1nCA rule).

We call One Neighbor Binary Cellular Automata the cellular automata $(\mathcal{L}, \mathcal{S}, \mathcal{N}, \mathcal{F})$ where

- $\mathcal{L} = [c_0, c_1, \dots, c_n]$ is an array of n cells,
- $\mathcal{S} = \{0, 1\}$ is the set of states ($k = 2$),
- $\mathcal{N}_c$ is neighborhood of the cell c and $\forall c : \mathcal{L} \quad |\mathcal{N}_c| = 2$,
- $f : \mathcal{S}^2 \to \mathcal{S}$ is a transition function.

Denoting the cell c at position i as c_i, the neighborhood $\mathcal{N}_{c_i}^{(t)}$ of the cell c_i at time t is defined as

$$\mathcal{N}_{c_i}^{(t)} = [c_i, n_{c_i}^{(t)}] \tag{4.13}$$

where $n_{c_i}^{(t)}$ is the neighbor of the cell, given by

$$n_{c_i}^{(t)} = \begin{cases} c_{i+1} & \text{if } t \text{ is even} \\ c_{i-1} & \text{otherwise} \end{cases} \tag{4.14}$$

The neighborhood of the cell c_i at different time steps is shown in Figure 4.10.

Following the Wolfram's notation, the rules are characterized by a sequence of binary values ($\beta_i \in S$) associated with each of the 4 possible patterns for the neighborhood. The transition function is defined as:

$$f(c_i, n_{c_i}^{(t)}) = \begin{cases} \beta_0 & \text{if } c_i = 0, n_{c_i}^{(t)} = 0 \\ \beta_1 & \text{if } c_i = 0, n_{c_i}^{(t)} = 1 \\ \beta_2 & \text{if } c_i = 1, n_{c_i}^{(t)} = 0 \\ \beta_3 & \text{if } c_i = 1, n_{c_i}^{(t)} = 1 \end{cases} \tag{4.15}$$

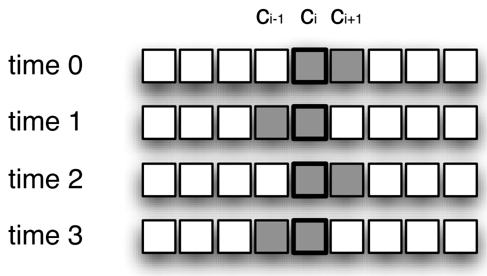


Figure 4.10: *The neighborhood of the cell c_i at different time steps.*

As shown in Figure 4.11, there are 16 possible transition functions, identified by a rule number R computed as

$$R = \sum_{i=0}^{3} \beta_i 2^i \tag{4.16}$$

For example, the "Rule $6\triangle$" is characterized by the following transition function:

$$f(c_i, n_{c_i}^{(t)}) = \begin{cases} 0 & \text{if } c_i = 0, n_{c_i}^{(t)} = 0 \\ 1 & \text{if } c_i = 0, n_{c_i}^{(t)} = 1 \\ 1 & \text{if } c_i = 1, n_{c_i}^{(t)} = 0 \\ 0 & \text{if } c_i = 1, n_{c_i}^{(t)} = 1 \end{cases} \tag{4.17}$$

Another possibility for representing the rules, is considered each rule as a boolean function. Considering 0 as $\perp$ and 1 as $\top$, each rule can be rewritten as:

Rule $0\triangle$	$f(c_i, n_{c_i}^{(t)}) = \perp$	Rule $8\triangle$	$f(c_i, n_{c_i}^{(t)}) = c_i \wedge n_{c_i}^{(t)}$
Rule $1\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg(c_i \vee n_{c_i}^{(t)})$	Rule $9\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg(c_i \oplus n_{c_i}^{(t)})$
Rule $2\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg c_i \wedge n_{c_i}^{(t)}$	Rule $10\triangle$	$f(c_i, n_{c_i}^{(t)}) = n_{c_i}^{(t)}$
Rule $3\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg c_i$	Rule $11\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg(c_i \wedge \neg n_{c_i}^{(t)})$
Rule $4\triangle$	$f(c_i, n_{c_i}^{(t)}) = c_i \wedge \neg n_{c_i}^{(t)}$	Rule $12\triangle$	$f(c_i, n_{c_i}^{(t)}) = c_i$
Rule $5\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg n_{c_i}^{(t)}$	Rule $13\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg(\neg c_i \wedge n_{c_i}^{(t)})$
Rule $6\triangle$	$f(c_i, n_{c_i}^{(t)}) = c_i \oplus n_{c_i}^{(t)}$	Rule $14\triangle$	$f(c_i, n_{c_i}^{(t)}) = c_i \vee n_{c_i}^{(t)}$
Rule $7\triangle$	$f(c_i, n_{c_i}^{(t)}) = \neg(c_i \wedge n_{c_i}^{(t)})$	Rule $15\triangle$	$f(c_i, n_{c_i}^{(t)}) = \top$

The configuration of a cellular automata is a mapping $q : \mathcal{L} \to \mathcal{S}$ which assigns to each cell of the array $\mathcal{L}$ a state from $\mathcal{S}$. We denoted with q_t the configuration of a cellular automata at time t:

$$q_t = [s_0, s_1, \ldots, s_n] \in \mathcal{S}^n \tag{4.18}$$

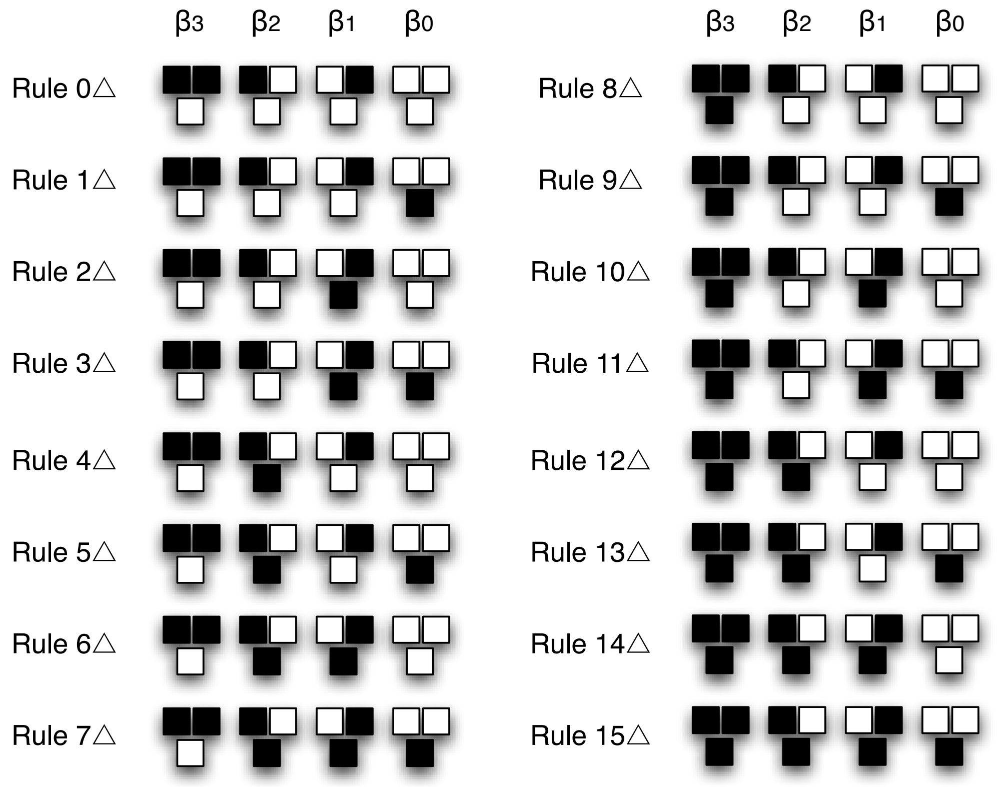


Figure 4.11: *Representation of the 16 One Neighbor Binary Cellular Automata transition rules.*

where n is the number of cells of $\mathcal{L}$. Given an initial configuration q_0, the evolution of an automaton is represented by a sequence of configurations:

$$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_t \tag{4.19}$$

A deterministic finite cellular automaton eventually falls into a cycle (with period $p > 1$) or a fixed point ($p = 1$):

$$\begin{aligned} q_t \rightarrow q_{t+1} \rightarrow q_{t+2} \rightarrow \ldots \rightarrow q_{t+p} \\ q_t = q_{t+p} \\ q_{t+1} = q_{t+p+1} \\ q_{t+2} = q_{t+p+2} \\ \vdots \\ q_{t+p} = q_{t+2p} \end{aligned} \tag{4.20}$$

We defined two constant configurations $\overline{0}$ and $\overline{1}$ as:

$$\overline{0} = [0, 0, \ldots, 0] \in \mathcal{S}^n \tag{4.21a}$$
$$\overline{1} = [1, 1, \ldots, 1] \in \mathcal{S}^n \tag{4.21b}$$

In the following section is presented ad classification of the 1nCA Rules. A central issue in the theory of cellular automata is the classification, i.e. understand how cellular automata can be meaningfully grouped according to their structure and behavior. There are mainly two approach for the classification of the cellular automata: the direct way, called Phenotypic Classification, to classified cellular automata is to observe their behavior through the spatial-temporal patterns they generates out of several random initial conditions, and then to use statistical metrics to quantify the observed behavior (Li et al., 1990a). Another approach, called Genotypic Classification, is based on the analysis of the automaton transition rules.

There are several works (e.g. (Wolfram, 1983a; Gutowitz et al., 1987; Li and Packard, 1990; Sutner, 1990; Wuensche, 1999)) focusing on the classification of the one dimensional cellular automata and in particular on the Elementary Cellular Automata. In this section we present an approach of genotypic classification applied to the One Neighbor Binary Cellular Automata. The idea of a genotypic classification of cellular automata is to divide a population of automata into groups according to the intrinsic properties of the rules. The aim is that some features of the cellular automata behaviors are predictable on the basis of a genotypic classification.

4.2.1 Totalistic Rules

Half of the possible rules are *totalistic*. A cellular automaton is called *totalistic* if the value of a cell depends only on the sum of the values of its neighbors at the previous time step, and not on their individual values (Wolfram, 1983b). The sum n of the neighborhood cells is computed $n = c_i + n_{c_i}^{(t)}$ and $0 \leq n \leq 2$. The following rules are totalistic:

Rule 0△ $\quad f(n) = 0$

Rule 8△ $$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \end{cases}$$

Rule 1△ $$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \\ 0 & \text{if } n = 2 \end{cases}$$

Rule 9△ $$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \end{cases}$$

Rule 6△ $f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 2 \end{cases}$ Rule 14△ $f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \end{cases}$

Rule 7△ $f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 2 \end{cases}$ Rule 15△ $f(n) = 1$

4.2.2 Neighbor-Independent and Self-Independent

A rule is *Neighbor-Independent* if the value of a cell depends only on its previous value and not on the value of the neighbors. Formally, a rule is *Neighbor-Independent* if

$$\forall s \in \mathcal{S}, f(s, 0) = f(s, 1) \tag{4.22}$$

so, according to the definition of the transaction function, a rule is *Neighbor-Independent* if

$$\beta_0 = \beta_1, \beta_2 = \beta_3 \tag{4.23}$$

A rule is *Self-Independent* if the value of a cell depends only on the value of the neighbors and not on its previous value. Formally, a rule is *Self-Independent* if

$$\forall s \in \mathcal{S}, f(0, s) = f(1, s) \tag{4.24}$$

so, according to the definition of the transaction function, a rule is *Self-Independent* if

$$\beta_0 = \beta_2, \beta_1 = \beta_3 \tag{4.25}$$

The *Neighbor-Independent* and *Self-Independent* rules are shown in Table 4.1.

Table 4.1: Neighbor-Independent *and* Self-Independent *rules*

Rule	Ind.		Rule	Ind.		Rule	Ind.		Rule	Ind.	
Rule 0△	N	S	Rule 1△			Rule 2△			Rule 3△	N	
Rule 4△			Rule 5△		S	Rule 6△			Rule 7△		
Rule 8△			Rule 9△			Rule 10△		S	Rule 11△		
Rule 12△	N		Rule 13△			Rule 14△			Rule 15△	N	S

4.2.3 λ-parameter

An even cruder piece of information about a rule is the number of non-quiescent outputs in a rule-table. For the One Neighbor Binary Cellular Automata, this parameter is equal to the number of β parameters that are equal to one and can be calculated as

$$c = \sum_{i=0}^{3} \beta_i \tag{4.26}$$

Langton (Langton, 1990) proposed the so called λ-parameter as an order-chaos parameter for Cellular Automata. This parameter measures the density of non-quiescent (not zero) outputs in a rule-table. For the One Neighbor Binary Cellular Automata the λ-parameter can be calculated as:

$$\lambda = \frac{c}{k^n} = \frac{1}{4}\sum_{i=0}^{3} \beta_i \tag{4.27}$$

where k is the number of states and n is the neighborhood size. λ varies between 0 (order) to 0.5 (chaos) to 1 (order). As λ is increased from 0 to 0.5 (or decreased from 1 to 0.5), the automata move from having the most homogeneous rule tables to having the most heterogeneous. The values of the λ parameter for all the rules are shown in Table 4.2.

Table 4.2: *The rules classified according to the λ parameter*

$\lambda = 0$	Rule $0\triangle$					
$\lambda = 0.25$	Rule $1\triangle$	Rule $2\triangle$	Rule $4\triangle$	Rule $8\triangle$		
$\lambda = 0.5$	Rule $3\triangle$	Rule $5\triangle$	Rule $6\triangle$	Rule $9\triangle$	Rule $10\triangle$	Rule $12\triangle$
$\lambda = 0.75$	Rule $7\triangle$	Rule $11\triangle$	Rule $13\triangle$	Rule $14\triangle$		
$\lambda = 1$	Rule $15\triangle$					

Langton presented evidence that there is some correlation between the λ parameter and the behavior of an "average" Cellular Automata on an "average" initial conguration (Langton, 1990). Behavior was characterized in terms of quantities such as single-site entropy, two-site mutual information, dierence-pattern spreading rate, and average transient length. Generally the correlation is quite good for very low and very high $lamda$ values, which predict fixed-point or short-period behavior. However, for intermediate λ values, there is a large degree of variation in behavior (Mitchell et al., 1993).

4.2.4 Sensitivity

Binder (1994, 1993) proposed the *sensitivity parameter* μ, motivated by the observation that the Wolfram classes are characterized by its sensitivity to changes in the state of a unique cell of the neighborhood of the transition rule.

Sensitivity is dened as the number of changes in the outputs of the transition rule, caused by changing the state of each cell of the neighborhood, one cell at a time, over all possible neighborhoods of the rule being considered:

$$\mu = \frac{1}{nm} \sum_{n} \sum_{j=1}^{m} \frac{\delta f}{\delta s_j} \tag{4.28}$$

where m is the number of cells in the neighborhood and n is the number of possible neighborhoods in the rule table. For 1 Neighbor Cellular Automata, $m = 2$, and $n = 2^m = 4$. The Boolean derivate for Cellular Automata (Vichniac, 1990) $\frac{\delta f}{\delta s_j}$ is equal to 1 if $f(s_1, \ldots, s_j, \ldots) \neq f(s_1, \ldots, \neg s_j, \ldots)$, otherwise is equal to 0.

Table 4.3: *The values of the* sensitivity parameter *for all the rules*

Rule	μ	Rule	μ	Rule	μ	Rule	μ
Rule 0$\triangle$	0	Rule 1$\triangle$	0.5	Rule 2$\triangle$	0.5	Rule 3$\triangle$	0.5
Rule 4$\triangle$	0.5	Rule 5$\triangle$	0.5	Rule 6$\triangle$	1	Rule 7$\triangle$	0.5
Rule 8$\triangle$	0.5	Rule 9$\triangle$	1	Rule 10$\triangle$	0.5	Rule 11$\triangle$	0.5
Rule 12$\triangle$	0.5	Rule 13$\triangle$	0.5	Rule 14$\triangle$	0.5	Rule 15$\triangle$	0

The values of the sensitivity parameter for all the rules is presented in Table 4.3. The sensitivity parameter takes on three different values: 0, 0.5, and 1. The sensitivity parameter helps to relatively discriminate null and chaotic behaviors: the null behavior happens in rules with low sensitivity and the chaotic behavior happens in rules with high sensitivity. Fixed-point and periodic behaviors are concentrated around 0.5.

4.2.5 Rule density

The Rule density is a simply parameter introduced to describe the rules behavior. The rule density, $R\rho$, is computed as:

$$R\rho = (\lambda - \frac{1}{2})\, 2^{(\beta_3 - \beta_0)} + \frac{1}{2} \tag{4.29}$$

Roughly speaking, the rule density indicates average fraction of sites with value equal to one in the rule dynamic evolution. The rule density value is comprised between zero and one. A value of zero indicated that a rule converges (for most of the initial configurations) to zero state in all the cells, a value of one indicated a convergence to one. The rule density values for all the rules are shown in Table 4.7.

Table 4.4: *The values of the* rule density *for all the rules*

Rule	$R\rho$	Rule	$R\rho$	Rule	$R\rho$	Rule	$R\rho$
Rule 0△	0	Rule 1△	0.375	Rule 2△	0.25	Rule 3△	0.5
Rule 4△	0.25	Rule 5△	0.5	Rule 6△	0.5	Rule 7△	0.625
Rule 8△	0	Rule 9△	0.5	Rule 10△	0.5	Rule 11△	0.75
Rule 12△	0.5	Rule 13△	0.75	Rule 14△	1	Rule 15△	1

4.2.6 Rules symmetries

One means of verication of the consistence of the rule density parameter (and also the other parameters) is the use of symmetries: if two rules are *conjugate*, the rule density of one rule is equals to $1 - R\rho$ of the other rule.

In Fatès (2003) the author defines the reflected, conjugate and reflected conjugate symmetries for the Elementary Cellular Automata. The only possible symmetry for the 1nCA is For any transition rule f, we can associate

f^*, the conjugate rule of f, defined by

$$\forall (c_i, n_{c_i}) \in \mathcal{S}^2, f^*(c_i, n_{c_i}) = f(\neg c_l, \neg n_{c_i}) \tag{4.30}$$

where $\neg$ denotes the operation of changing the zeros into ones and ones into zeros. The β^* parameters of the conjugate rule are calculated as:

$$\beta_3^* = \neg\beta_0 \quad \beta_2^* = \neg\beta_1 \quad \beta_1^* = \neg\beta_2 \quad \beta_0^* = \neg\beta_3 \tag{4.31}$$

In Table 4.5 are shown, for each rule, the reflected, the conjugated and the reflected conjugated rules. We can identified 6 classes of rules, shown in Table 4.6, according to the symmetries: we can group in one class all the rules that are symmetric (reflected, conjugated or reflected conjugated). The classes are named according to the lowest member index. Each class is formed by all totalistic or all not-totalistic rules.

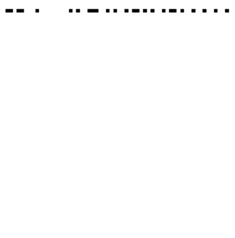
This kind of classification of the One Neighbor Binary Cellular Automata is important because we can restrict the study of the dynamic behavior to only one member of each class and the behavior of the other members can be simply inferred according to the symmetric relations.

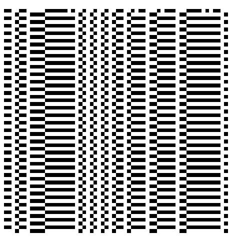
Table 4.5: *Rules Symmetries*

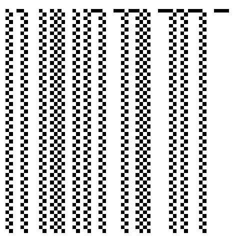
Rule		Conjugated rule		Rule		Conjugated rule	
Rule $0\triangle$	(0000)	Rule $15\triangle$	(1111)	Rule $1\triangle$	(0001)	Rule $7\triangle$	(0111)
Rule $2\triangle$	(0010)	Rule $11\triangle$	(1011)	Rule $3\triangle$	(0011)		
Rule $4\triangle$	(0100)	Rule $13\triangle$	(1101)	Rule $5\triangle$	(0101)		
Rule $6\triangle$	(0110)	Rule $9\triangle$	(1001)	Rule $7\triangle$	(0111)	Rule $1\triangle$	(0001)
Rule $8\triangle$	(1000)	Rule $14\triangle$	(1110)	Rule $9\triangle$	(1001)	Rule $6\triangle$	(0110)
Rule $10\triangle$	(1010)			Rule $11\triangle$	(1011)	Rule $2\triangle$	(0010
Rule $12\triangle$	(1100)			Rule $13\triangle$	(1101)	Rule $4\triangle$	(0100)
Rule $14\triangle$	(1110)	Rule $8\triangle$	(1000)	Rule $15\triangle$	(1111)	Rule $0\triangle$	(0000)

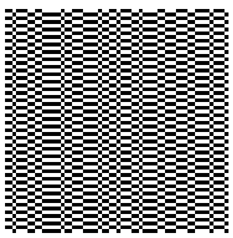
Table 4.6: *The rules divided according to the symmetries. The value of* rule density *is reported for each rule. The classes marked with* **T** *are formed by totalistic rules,* **N** *by Neighbor-Independent rules, and* **S** *by Self-Independent rules.*

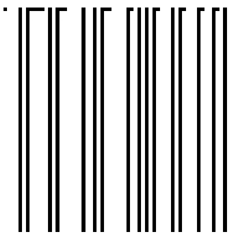
Class 0△TNS :	Rule $0\triangle$	$(R\rho = 0)$	Rule $15\triangle$	$(R\rho = 1)$
Class 1△T :	Rule $1\triangle$	$(R\rho = 0.375)$	Rule $7\triangle$	$(R\rho = 0.625)$
Class 2△ :	Rule $2\triangle$	$(R\rho = 0.25)$	Rule $11\triangle$	$(R\rho = 0.75)$
Class 3△N :	Rule $3\triangle$	$(R\rho = 0.5)$		
Class 4△ :	Rule $4\triangle$	$(R\rho = 0.25)$	Rule $13\triangle$	$(R\rho = 0.75)$
Class 5△S :	Rule $5\triangle$	$(R\rho = 0.5)$		
Class 6△T :	Rule $6\triangle$	$(R\rho = 0.5)$	Rule $9\triangle$	$(R\rho = 0.5)$
Class 8△T :	Rule $8\triangle$	$(R\rho = 0)$	Rule $14\triangle$	$(R\rho = 1)$
Class 10△S :	Rule $10\triangle$	$(R\rho = 0.5)$		
Class 12△N :	Rule $12\triangle$	$(R\rho = 0.5)$		

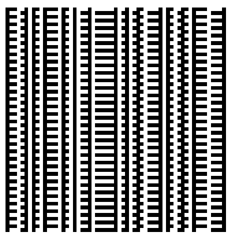

Rule 0△ (Class 0△TNS)
$R\rho = 0$
$\lambda = 0, \mu = 0$

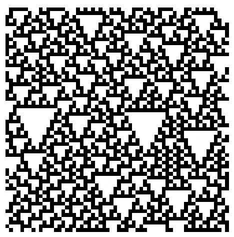

Rule 1△ (Class 1△T)
$R\rho = 0.375$
$\lambda = 0.25, \mu = 0.5$

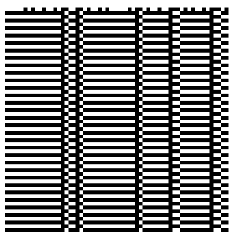

Rule 2△ (Class 2△)
$R\rho = 0.25$
$\lambda = 0.25, \mu = 0.5$

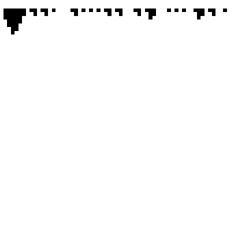

Rule 3△ (Class 3△N)
$R\rho = 0.5$
$\lambda = 0.5, \mu = 0.5$

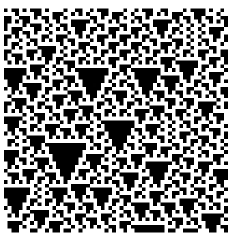

Rule 4△ (Class 4△)
$R\rho = 0.25$
$\lambda = 0.25, \mu = 0.5$

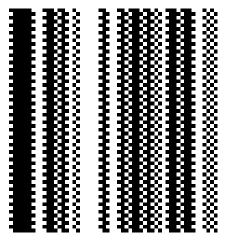

Rule 5△ (Class 5△TS)
$R\rho = 0.5$
$\lambda = 0.5, \mu = 0.5$

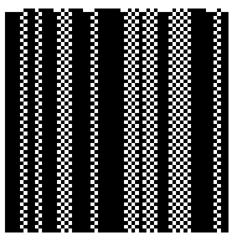

Rule 6△ (Class 6△T)
$R\rho = 0.5$
$\lambda = 0.5, \mu = 1$

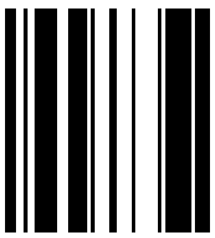

Rule 7△ (Class 1△T)
$R\rho = 0.625$
$\lambda = 0.75, \mu = 0.5$

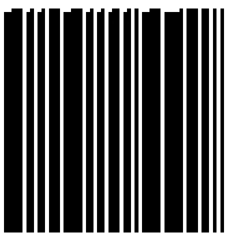

Rule 8△ (Class 8△T)
$R\rho = 0$
$\lambda = 0.25, \mu = 0.5$

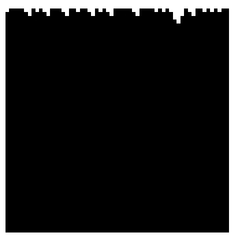

Rule 9△ (Class 6△T)
$R\rho = 0.5$
$\lambda = 0.5, \mu = 1$


Rule 10△ (Class 10△S)
$R\rho = 0.5$
$\lambda = 0.5, \mu = 0.5$


Rule 11△ (Class 2△)
$R\rho = 0.75$
$\lambda = 0.75, \mu = 0.5$


Rule 12△ (Class 12△N)
$R\rho = 0.5$
$\lambda = 0.5, \mu = 0.5$


Rule 13△ (Class 4△)
$R\rho = 0.75$
$\lambda = 0.75, \mu = 0.5$


Rule 14△ (Class 8△T)
$R\rho = 1$
$\lambda = 0.75, \mu = 0.5$

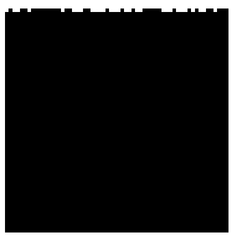

Rule 15△ (Class 0△T)
$R\rho = 1$
$\lambda = 1, \mu = 0$

Figure 4.12: *60 steps of the time evolution of all the 16 One Neighbor Binary Cellular Automata with the default synchronous update scheme and periodic boundaries conditions starting from an initial random configuration of 60 cells.*

Table 4.7: *Classification summary*

Rule	Bin	Wolfram Class	Li-Packard Class	Symmetry Class	Tot.	Ind.	λ	$R\rho$	μ
$0\triangle$	0000	W1	Null	$0\triangle$	T	NS	0	0	0
$1\triangle$	0001	W2	Two-Cycle	$1\triangle$	T		0.25	0.375	0.5
$2\triangle$	0010	W2	Two-Cycle	$2\triangle$			0.25	0.25	0.5
$3\triangle$	0011	W2	Two-Cycle	$3\triangle$		N	0.5	0.5	0.5
$4\triangle$	0100	W1	Fixed-Point	$4\triangle$			0.25	0.25	0.5
$5\triangle$	0101	W2	Two-Cycle	$5\triangle$		S	0.5	0.5	0.5
$6\triangle$	0110	W3	Chaotic	$6\triangle$	T		0.5	0.5	1
$7\triangle$	0111	W2	Two-Cycle	$1\triangle$	T		0.75	0.625	0.5
$8\triangle$	1000	W1	Null	$8\triangle$	T		0.25	0	0.5
$9\triangle$	1001	W3	Chaotic	$6\triangle$	T		0.5	0.5	1
$10\triangle$	1010	W2	Two-Cycle	$10\triangle$		S	0.5	0.5	0.5
$11\triangle$	1011	W2	Two-Cycle	$2\triangle$			0.75	0.75	0.5
$12\triangle$	1100	W2	Fixed-Point	$12\triangle$		N	0.5	0.5	0.5
$13\triangle$	1101	W2	Fixed-Point	$4\triangle$			0.75	0.75	0.5
$14\triangle$	1110	W1	Null	$8\triangle$	T		0.75	1	0.5
$15\triangle$	1111	W1	Null	$0\triangle$	T	NS	1	1	0

4.3 1nCA Spatiotemporal Patterns

In this section we present the effects of several update schemes on 1nCA automata dynamic evolutions.

In Figure 4.12 are shown the time evolution of all the 16 rules according with synchronous update scheme and periodic boundaries conditions, starting from an initial random configuration of 60 cells. As shown in the following sections, the different update schemes produce tremendous effects on the several automata.

The tested the following update schemes on all the 1nCA rules:

- Synchronous
- Random Cyclic
- Equal Frequency Clocked
- Random Order
- Random Independent

Moreover we tested the combined effect of the update scheme and the historic memory. We adopted the geometrically discounted memory mechanism proposed in Alonso-Sanz and Martín (2002); Alonso-Sanz and Martin (2003); Alonso-Sanz (2003, 2004, 2006, 2007); Alonso-Sanz and Cardenas (2008).

In the following paragraphs, we present the observation results for all the classes.

4.3.1 Class $0\triangle$TNS

The rules of this class (Rule $0\triangle$ and Rule $15\triangle$), that are Totalistic, Neighbor-Independent and Self-Independent, have a trivial behavior: they "erase" any initial configuration, reaching a fixed point. The limiting configuration (i.e., the automaton state of the entire lattice) is formed only by zeros for Rule $0\triangle$ ($R\rho = 0, \lambda = 0, \mu = 0$) the and ones for the Rule $15\triangle$ ($R\rho = 1, \lambda = 1, \mu = 0$). According to the Li-Packard classification, the rules of this class are classified as *Null rules* and as *Class 1* according to Wolfram.

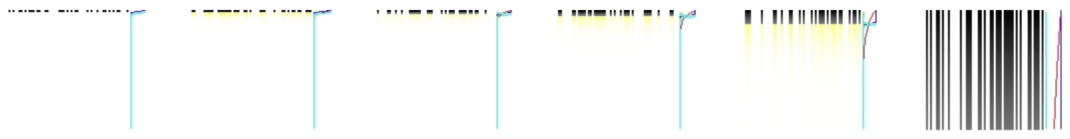


Figure 4.13: *Time space diagrams of Rule $0\triangle$ with different memory factor α.*

Since the rule are Neighbor-Independent, the behavior do not change varying the update strategy. The only parameter that affect the dynamic behavior is the memory factor. More formally, for the Rule $0\triangle$

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \overline{0} \tag{4.32}$$

and for the Rule $15\triangle$

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \overline{1} \tag{4.33}$$

where, t_t, the length of the transient. The value of t_t is equal to 1 for all the update strategy when the memory factor $\alpha \leq 0.5$. Increasing the memory factor, the dynamics converge slower to the fixed point. In Figure 4.13 are shown time space diagrams of Rule $0\triangle$ with different values of the memory factor α.

4.3.2 Class $1\triangle$T

The Rule $1\triangle$ ($R\rho = 0.375, \lambda = 0.25, \mu = 0.5$) and Rule $7\triangle$ ($R\rho = 0.625, \lambda = 0.75, \mu = 0.5$) are Totalistic. With Synchronous, Cyclic, and Equal Frequency Clocked update strategies, they transform any initial pattern in a new pattern that reappears every two time steps. The rules of this class are *Two-Cycle rules* according to the Li-Packard classification and *Class 2* according to the Wolfram classification.

Formally, the behavior can be described as

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \begin{cases} q_{t_{t+1}} & \text{if } t - t_t \text{ is odd} \\ q_{t_{t+2}} & \text{if } t - t_t \text{ is even} \end{cases} \tag{4.34}$$

The values of t_t depends both on the update strategy and the memory factor. A rule of this class with the synchronous update and without memory ($\alpha = 0$) reaches the cycle in one step ($t_t = 1$), so

$$\forall\, q_1 \in \mathcal{S}^n, t > 1, q_t = \begin{cases} q_2 & \text{if } t \text{ is even} \\ q_3 & \text{if } t \text{ is odd} \end{cases} \tag{4.35}$$

As shown in Figure 4.14, with the Cyclic and Equal Frequency Clocked update schemes, the dynamical evolution slowly converges to an attractor cycles of period 2. Increasing the memory factor α, the convergence is even slower.

In Figure 4.15 are presented the time space diagrams of the Rule$\triangle 1$ using different update schemes starting from the same initial configuration ($q_1 = \overline{01}$). As shown in the figure, the dynamic evolutions of the rules, with the Random Independent and Random Order update schemes, don't seem to converge to an attractor cycle, nor starting from an "ordered" pattern, nor with an high memory factor ($\alpha = 0.99$).

We can identified several phases of the dynamic update. Some phases are present with all the update schemes. The phases are clearly identifiable when the memory factor is higher, since the phases length is longer. The phase p_1 is the initial phase and is always present. In this phase, the initial configuration is preserved. The length of the phase depend on the memory factor. If the is no memory, the length of this phase is equal to one.

The second phase p_2 is the transient phase. For the Synchronous and the Cyclic, and Equal Frequency Clocked update scheme, this phase is an intermediate phase between the initial phase and the attractor cycle phase. The initial configuration is "scrambled" until an attractor cycle is reached. Increasing the memory factor, the transient phase presents an initial cyclic sub-phase. In this sub–phase, the same configuration is reproduced with a period of 2. As shown in Figure 4.15, this sub-phase could be very long with an high memory factor ($\alpha = 0.99$). For the synchronous update scheme, this sub–phase constitute the whole second phase. This sub–phase is present also with the Random Order update scheme.

The third phase p_3 is the attractor cycle phase. This phase is terminal, i.e. the dynamical evolution never exit from this phase. The length of the cycle is equal to 2. With the update schemes, the dynamical evolution never enters in this phase. For the synchronous update scheme, the attractor cycle is independent from the memory factor, so starting from the same configuration, with different memory factors, the same attractor cycle is reached. For the Cyclic and Equal Frequency Clocked update schemes, the configuration of the attractor cycle depends both on the initial configuration an the order of activation of the cells. So, with the same parameters, the same initial configuration could reach different attractor cycles.

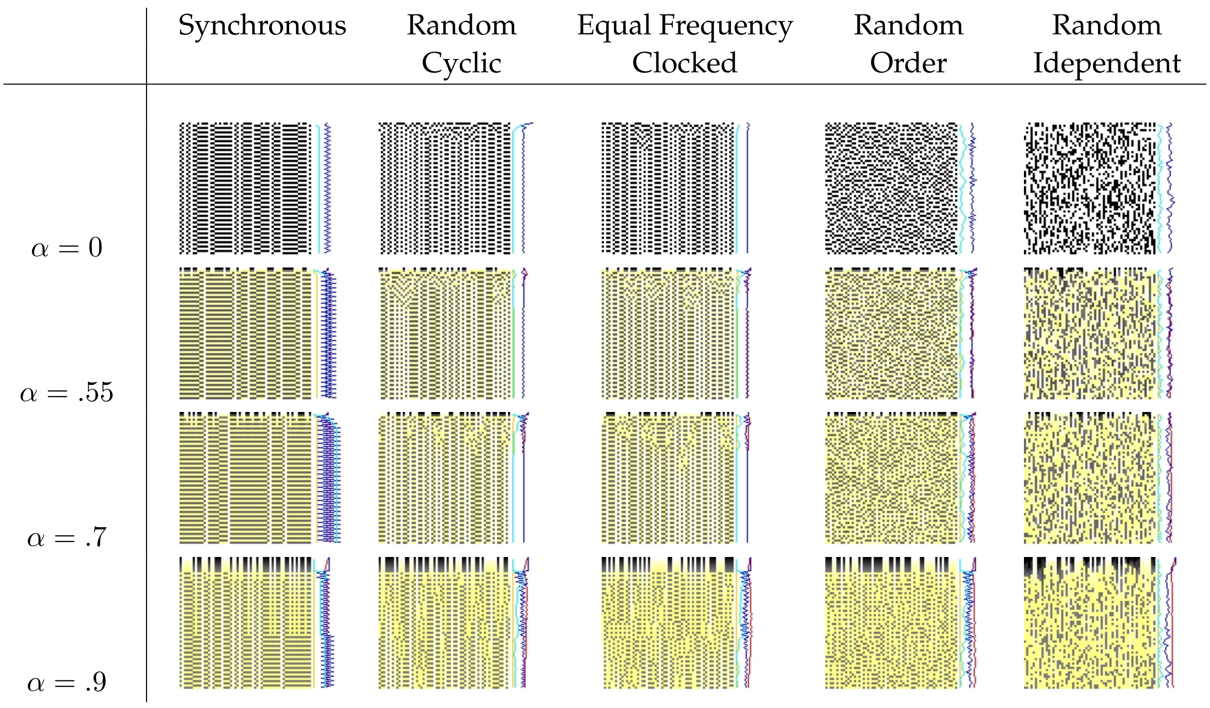


Figure 4.14: *Time space diagrams of Rule* $1\triangle$ *using different update schemes and memory factors.*

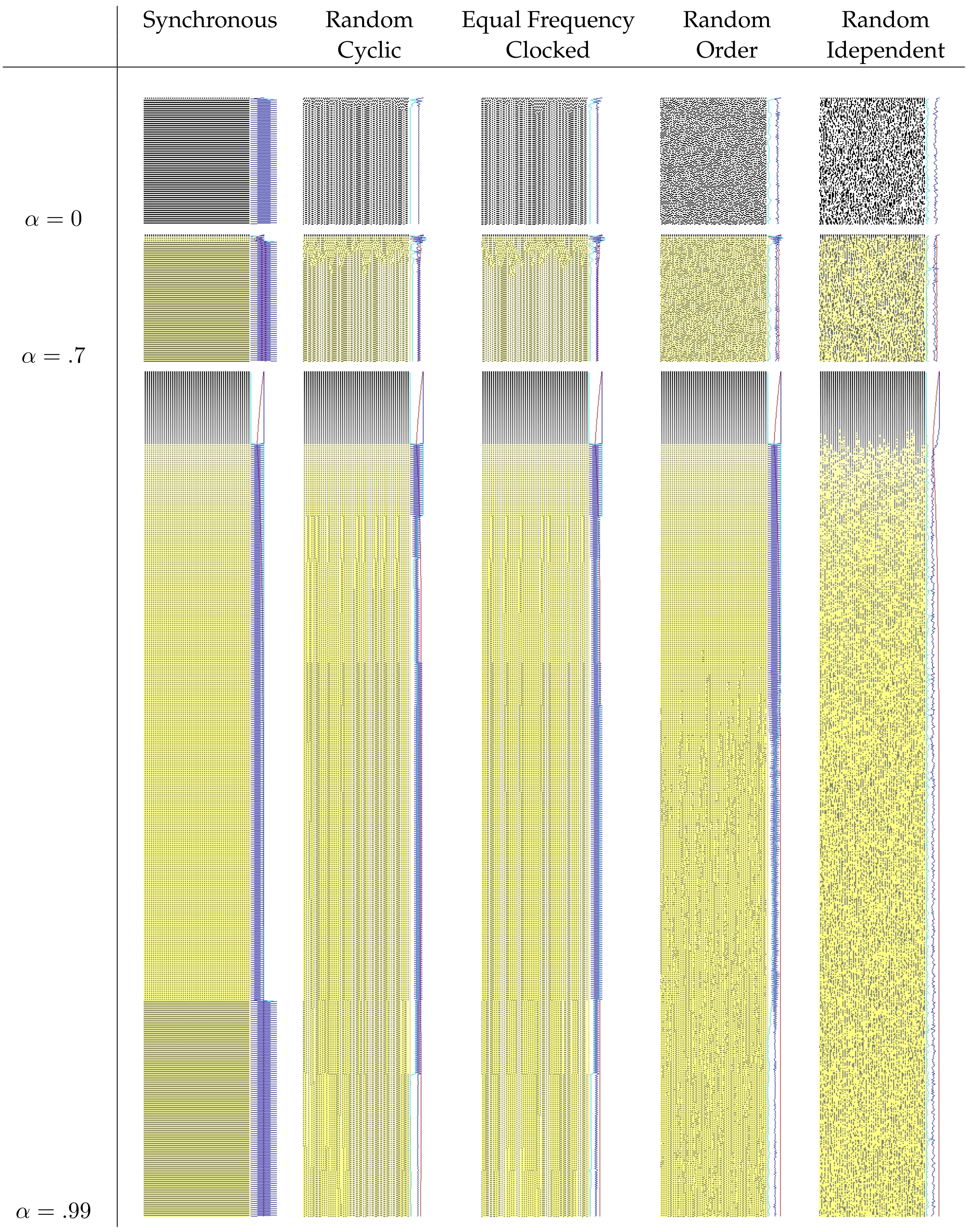


Figure 4.15: *Time space diagrams of Rule* $1\triangle$ *using different update schemes starting from the same initial configuration* ($q_1 = \overline{01}$).

4.3.3 Class $2\triangle$

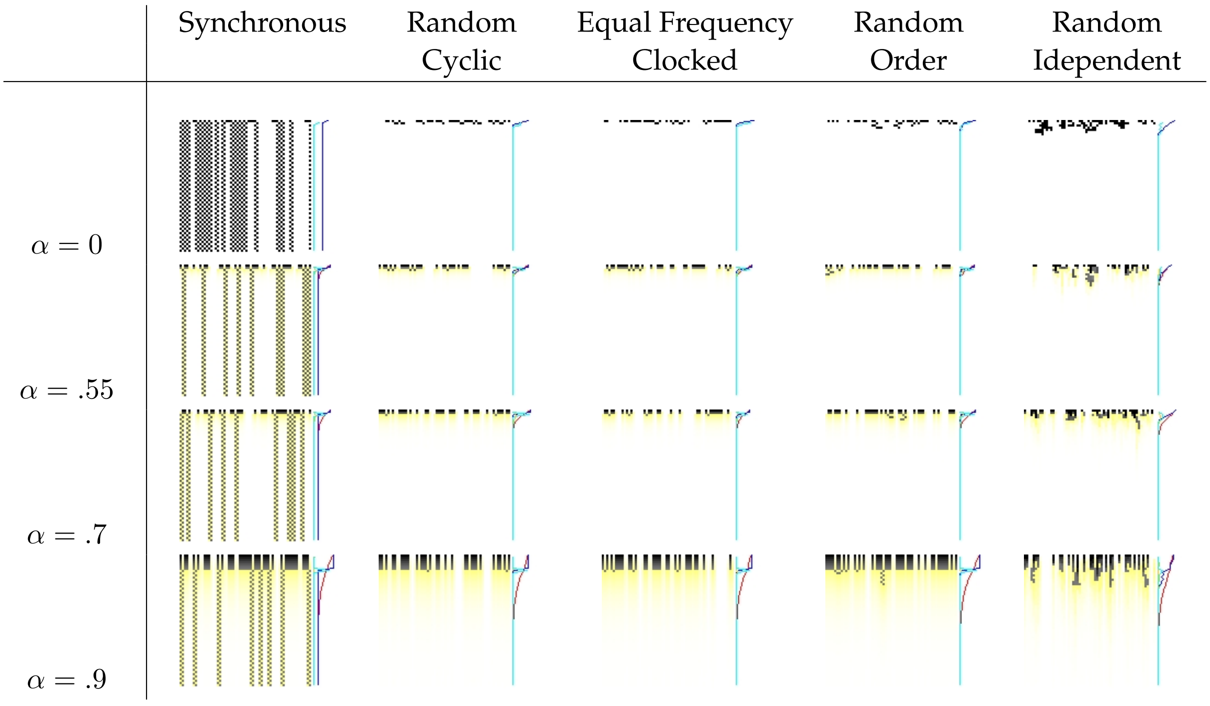


Figure 4.16: *Time space diagrams of Rule $2\triangle$ using different update schemes and memory factors.*

The rules of this class are Rule $2\triangle$ ($R\rho = 0.25, \lambda = 0.25, \mu = 0.5$) and Rule $11\triangle$ ($R\rho = 0.75, \lambda = 0.75, \mu = 0.5$). With synchronous update strategy, the rules transform any initial pattern in a new pattern that reappears every two time steps:

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \begin{cases} q_{t_t+1} & \text{if } t - t_t \text{ is odd} \\ q_{t_t+2} & \text{if } t - t_t \text{ is even} \end{cases} \tag{4.36}$$

The values of t_t depends only on the memory factor α. A rule of this class with the synchronous update and without memory ($\alpha = 0$) reaches the cycle in one step ($t_t = 1$). The rules of this class are *Two-Cycle rules* according to the Li-Packard classification and *Class 2* according to the Wolfram classification.

As show in 4.16, the rules have a different behavior with the asynchronous update schemes: starting from any initial configuration, the dynamic evolution finally reached the limiting configuration. The limiting configuration is formed only by zero for the Rule $2\triangle$:

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \overline{0} \tag{4.37}$$

and by ones for the Rule $11\triangle$:

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \overline{1} \tag{4.38}$$

The length of the transient t_t depends on the memory factor α.

4.3.4 Class $3\triangle$N

The Rule $3\triangle$ ($R\rho = 0.5, \lambda = 0.5, \mu = 0.5$) is a Neighbor-Independet rule. With Synchronous, Cyclic, Equal Frequency Clocked, and Random Order update strategies, the rule transforms any initial pattern into its complement. The pattern reappears every two time steps:

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \begin{cases} q_{t_{t+1}} & \text{if } t - t_t \text{ is odd} \\ \neg q_{t_{t+1}} & \text{if } t - t_t \text{ is even} \end{cases} \tag{4.39}$$

As show in 4.17, the length of the transient t_t depends on the memory factor α: without memory, t_t is equal to 0. The rules of this class are *Two-Cycle rules* according to the Li-Packard classification and *Class 2* according to the Wolfram classification. With the Random Independent update strategy, the rule has a chaotic behavior.

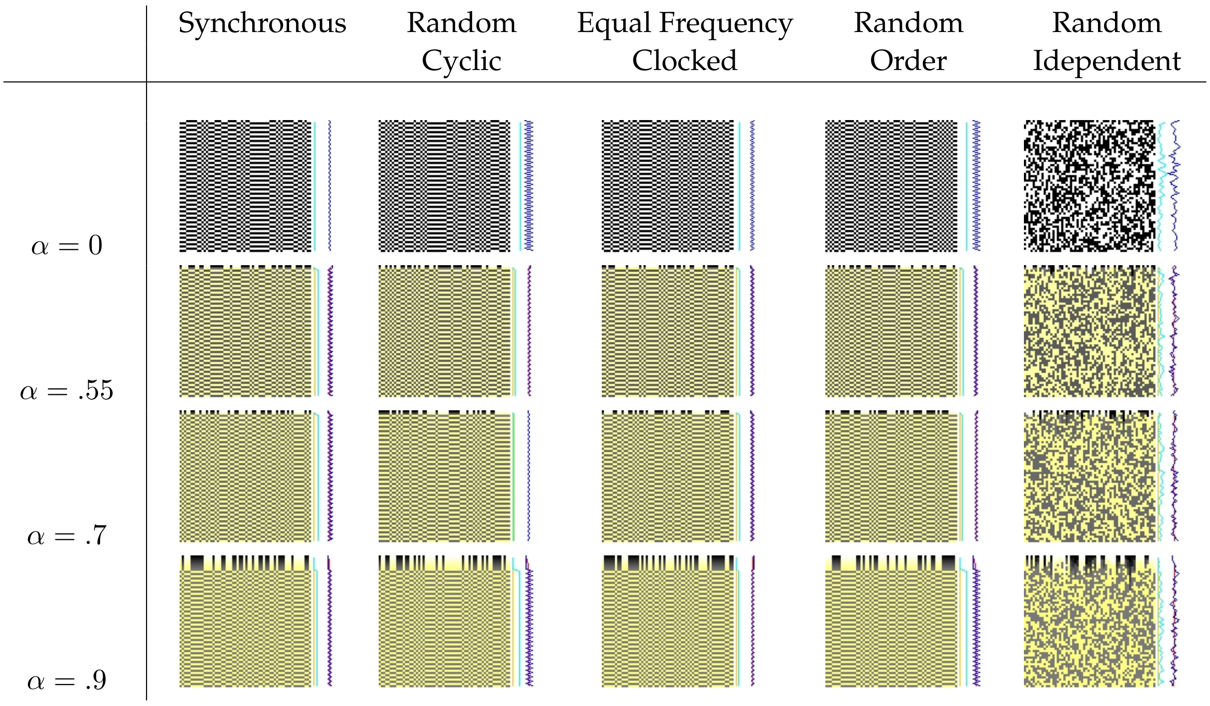


Figure 4.17: *Time space diagrams of Rule $3\triangle$ using different update schemes and memory factors.*

4.3.5 Class $4\triangle$

The rules of this class are the Rule $4\triangle$ ($R\rho = 0.25, \lambda = 0.25, \mu = 0.5$) and the Rule $13\triangle$ ($R\rho = 0.75, \lambda = 0.75, \mu = 0.5$). In Figure 4.18 are presented the time space diagrams of the Rule$\triangle$4 using different update schemes and memory factors. The rule transform any initial pattern into a new pattern that is the limiting configuration:

$$\forall\ q_1 \in \mathcal{S}^n, t > t_t, q_t = q_{t_t} \tag{4.40}$$

The length of the transient t_t depends on the memory factor α and the update strategy. For the synchronous update strategy, without memory, t_t is equal to 2. According to the Li-Packard classification, this rules is classified as *Fixed point rule* and as *Class 1* according to Wolfram.

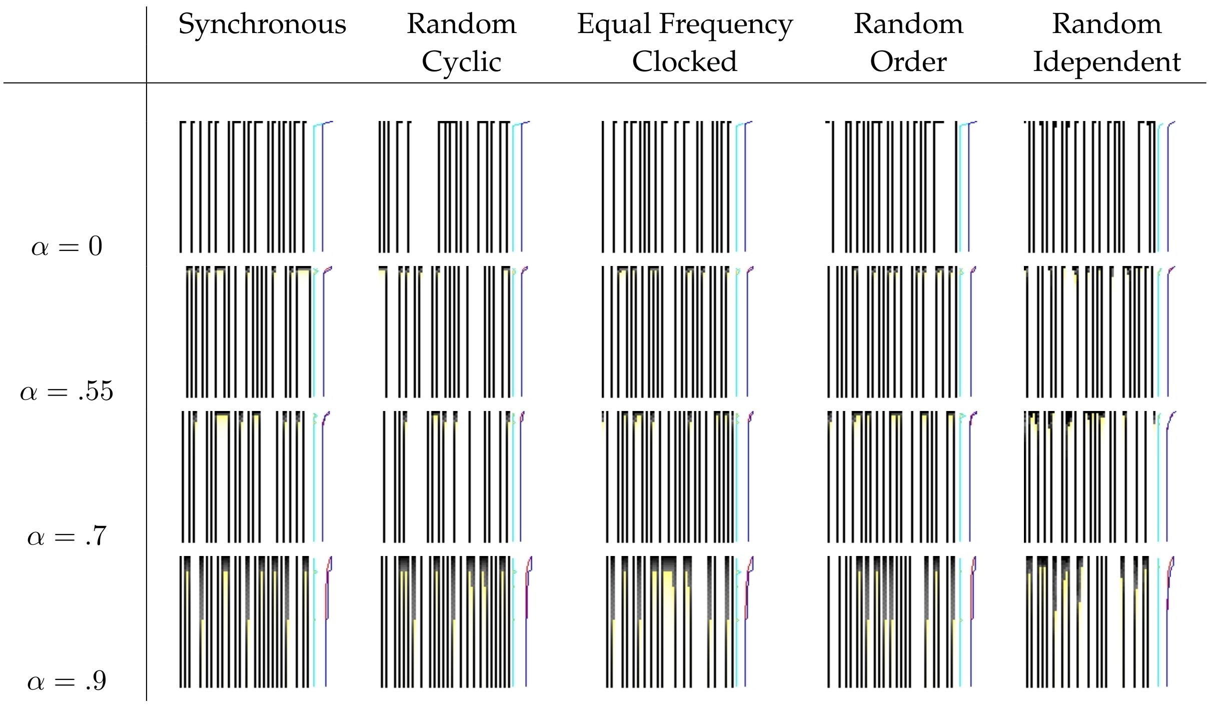


Figure 4.18: *Time space diagrams of Rule $4\triangle$ using different update schemes and memory factors.*

4.3.6 Class $5\triangle$S

The Rule $5\triangle$ ($R\rho = 0.5, \lambda = 0.5, \mu = 0.5$) is Self-independent, i.e. the next value depends only on the value of its neighbor. The rule has a periodic behavior: after a transient phase p_1, the dynamic behavior follows into an attractor cycle of period 2.

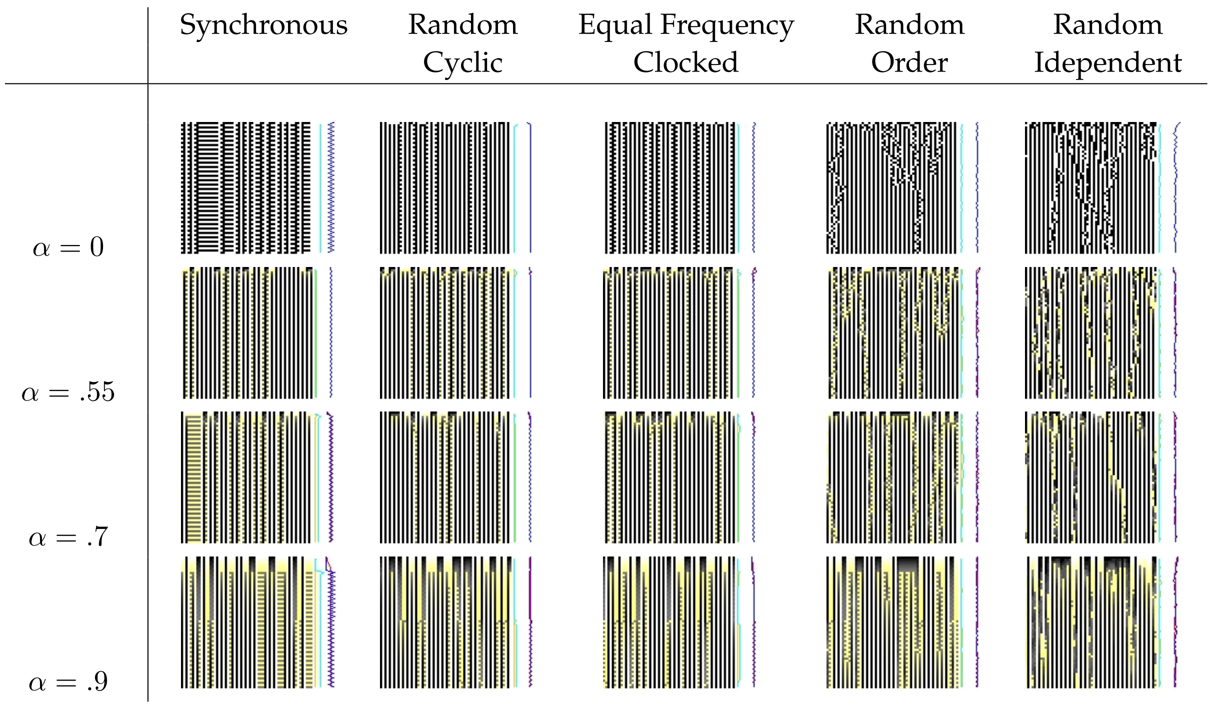


Figure 4.19: *Time space diagrams of Rule $5\triangle$ using different update schemes and memory factors.*

In Figure 4.19 are presented the time space diagrams of the Rule$\triangle$5 using different update schemes. The rule of this class is *Two-Cycle rules* according to the Li-Packard classification and *Class 2* according to the Wolfram classification. Formally, the behavior for Synchronous, Cyclic, and Clocked schemes, can be described as

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \begin{cases} q_{t_{t+1}} & \text{if } t - t_t \text{ is odd} \\ q_{t_{t+2}} & \text{if } t - t_t \text{ is even} \end{cases} \tag{4.41}$$

The length of the transient depends on several factors: the update scheme, the memory factor, the initial configuration and the boundaries conditions. For the synchronous update scheme, without memory, t_t is equal to 0. The length of the transient for the Cyclic, and Equal Frequency Clocked update schemes is little longer then for the synchronous one. Increasing the memory factor α, the length of the transient become longer.

The configuration $\overline{01}$ is invariant, so

$$q_1 = \overline{01}, t > 1, q_t = q_1 \tag{4.42}$$

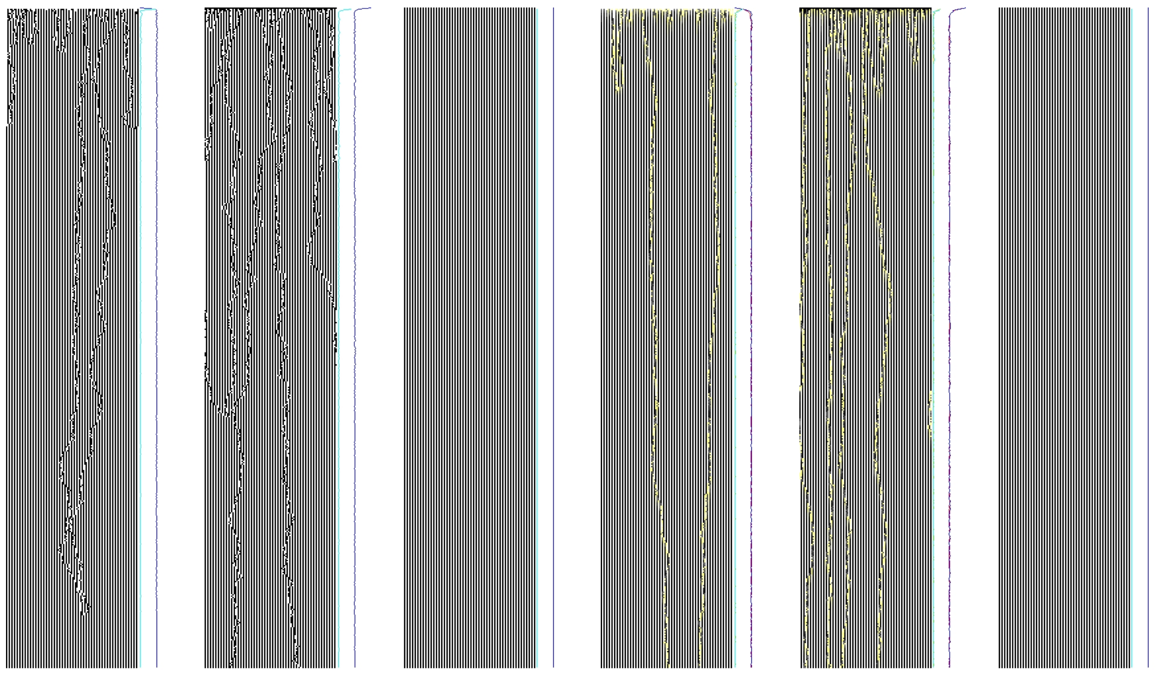

$\alpha = 0$ $q_1 = \overline{0}$
$\alpha = 0$ $q_1 = \overline{1}$
$\alpha = 0$ $q_1 = \overline{01}$
$\alpha = .7$ $q_1 = \overline{0}$
$\alpha = .7$ $q_1 = \overline{1}$
$\alpha = .7$ $q_1 = \overline{01}$

Figure 4.20: *Time space diagrams of Rule* $5\triangle$ *using the Random Idependent update scheme, starting from different initial configurations.*

The rule has a different behavior with the Random update schemes. As shown in Figure 4.20, the spaces is divided into region with the 01 pattern repeated over the region. The regions are divided by *edges* characterized by "broken" patterns 00 or 11, a shown in Figure 4.21.

Such regions grow, merge and collapse. Regions transformations are determined by edge movements. Transformations never occur in the middle of a region, i.e. a region cannot split into two regions. If a region grows enough to occupied all the spaces, that regions remains constant in the time (a fixed point is reached). The two fixed point are the configurations $\overline{01}$ and $\overline{10}$. With periodic boundaries, if the number of cells is odd, such fixed configurations can never be reached.

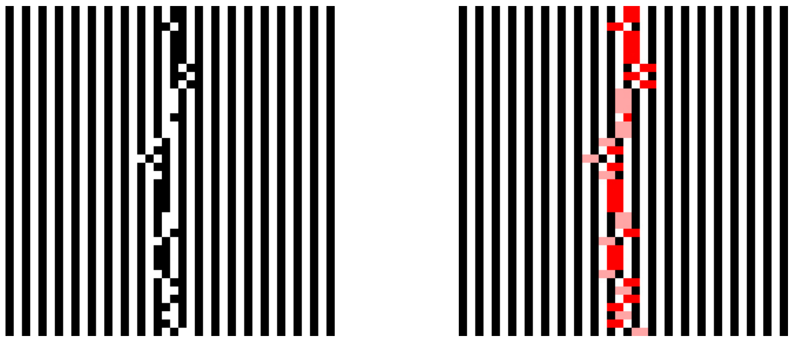


Figure 4.21: *Time space diagrams of Rule* $5\triangle$ *using the Random Idependent update scheme, showing the edge between two zone. In the image on the right, the edge is in evidence.*

4.3.7 Class $6\triangle$T

The rules of this class are Rule $6\triangle$ ($R\rho = 0.5, \lambda = 0.5, \mu = 1$) and Rule $9\triangle$ ($R\rho = 0.5, \lambda = 0.5, \mu = 1$). The dynamic behaviors of the Rule $6\triangle$ using different update schemes and memory factors are shown in Figure 4.23. The rules of this class are *Chaotic*: these rules are characterized by the exponential divergence of its cycle length with the system size, and for the instability with respect to perturbations. If the number of cells is finite, for the Synchronous, Random Cyclic, and Equal Frequency Clocked schemes, the evolution eventually falls into a cycle (with period $p > 1$) or a fixed point ($p = 1$). The configuration $\overline{0}$ is the fixed point of the Rule $6\triangle$, the configuration $\overline{1}$ is the fixed point of the Rule $9\triangle$. These configurations are fixed points also using the Random update schemes.

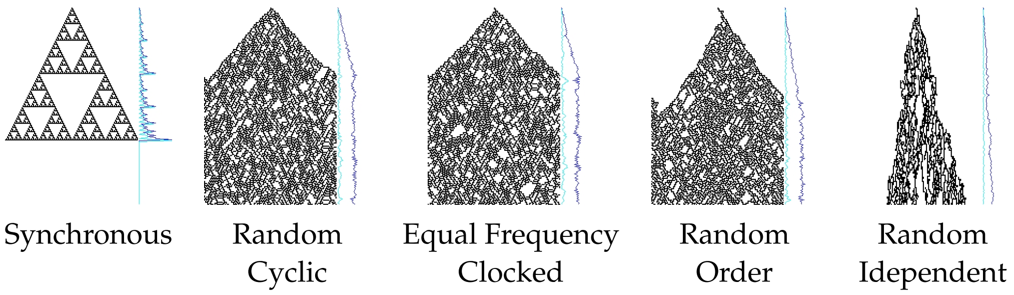


Figure 4.22: *Time space diagrams of Rule* $6\triangle$ *using different update schemes starting from a single seed.*

Changing update scheme has dramatic effect on the rule of this class. As shown in Figure 4.22, with the Synchronous update scheme, the Rule $6\triangle$ produces a dynamic evolution similar to the *Sierpinski Triangle* fractal. This typical shape is not present with any of the other update schemes.

Moreover if the automaton has periodic boundaries conditions and the number of cells is a power of two, starting from an initial configuration, the evolution of the synchronous automata eventually reaches the fixed point. The automata with the other update schemes does not have this behavior.

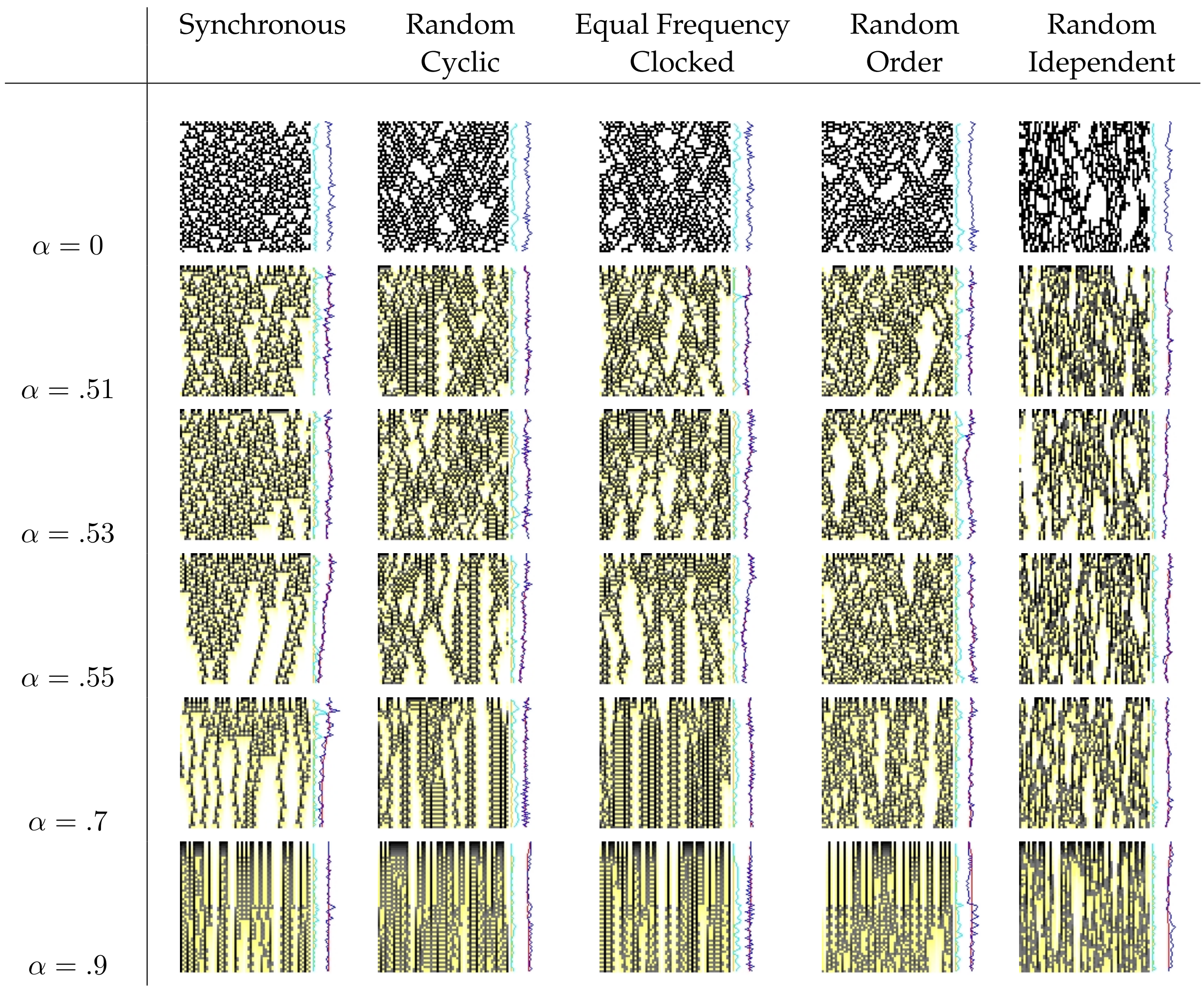


Figure 4.23: *Time space diagrams of Rule* $6\triangle$ *using different update schemes and memory factors.*

4.3.8 Class $8\triangle$T

The rules of this class (Rule $8\triangle$ and Rule $14\triangle$) reach a fixed point for almost any initial configuration. As shown in Figures 4.24 and 4.25, the limiting configuration, formed only by zeros for Rule $0\triangle$ ($R\rho = 8$) the and ones for the Rule $14\triangle$ ($R\rho = 1$), is reached after a number of steps depending on the max number of consecutive ones (for the Rule $8\triangle$) or zeros (Rule $14\triangle$). If the initial state is composed only of ones (for Rule $8\triangle$) or zeros (for Rule $14\triangle$) the automata maintain this configurations as fixed point. Formally, for the Rule $8\triangle$:

$$q_t = \begin{cases} \overline{0} & \text{if } q_0 \neq \overline{1}, t > t_t \\ \overline{1} & \text{if } q_0 = \overline{1}, t > 1 \end{cases} \tag{4.43}$$

The rules of this class are *Null rules* according to the Li-Packard classification and *Class 1* according to the Wolfram classification.

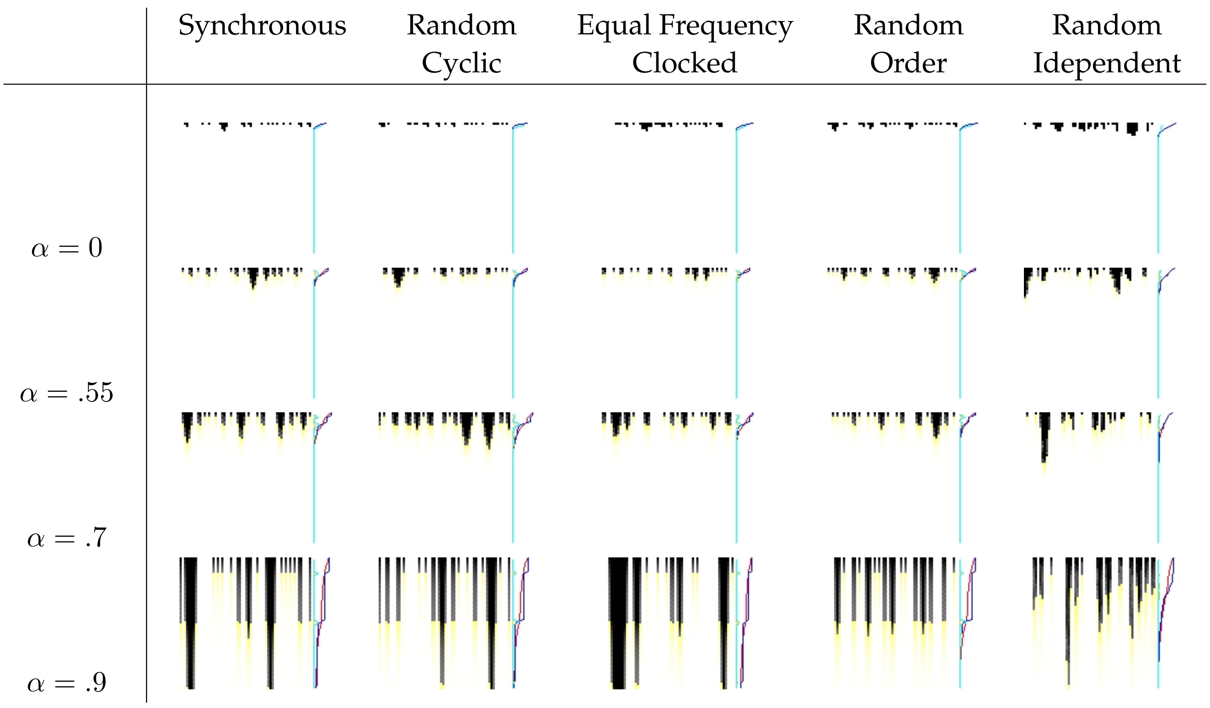


Figure 4.24: *Time space diagrams of Rule $8\triangle$ using different update schemes and memory factors.*

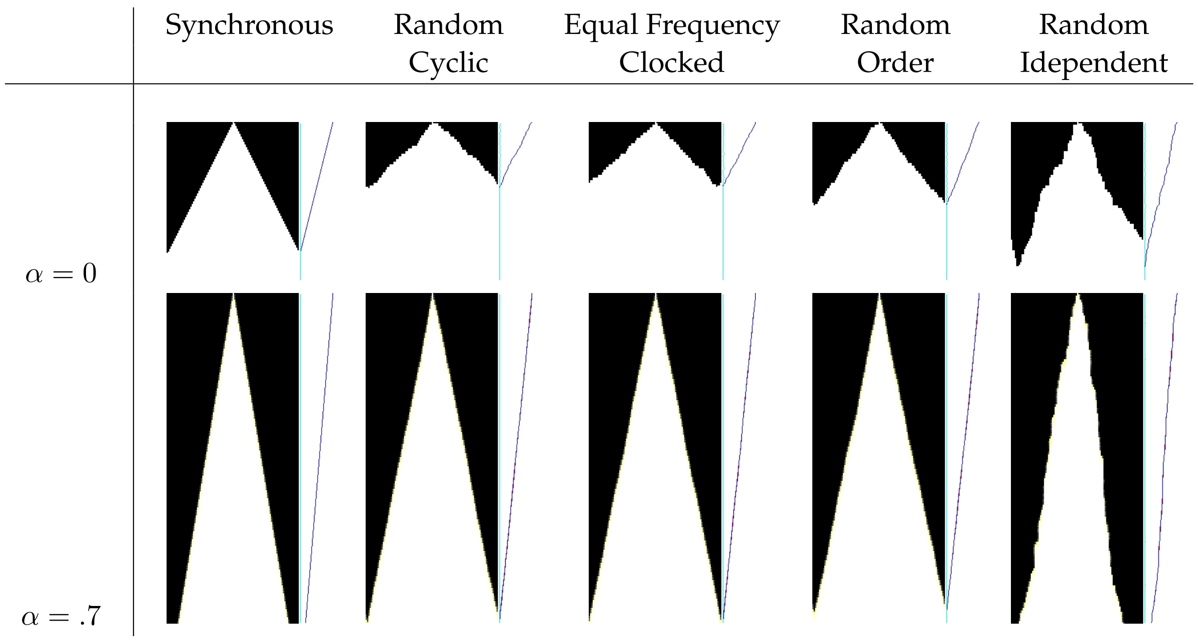


Figure 4.25: *Time space diagrams of Rule $8\triangle$ using different update schemes and memory factors, starting from the same initial configuration (all cells with state equals to 1 except one).*

4.3.9 Class $10\triangle$S

The Rule $10\triangle$ ($R\rho = 0.5, \lambda = 0.5, \mu = 0.5$) is Self-independent, i.e. the next value depends only on the value of its neighbor. The rule has a periodic behavior: after a transient phase p_1, the dynamic behavior follows into an attractor cycle of period 2. The length of the transient phase depends on the memory factor, the initial pattern, and the update scheme. In Figure 4.26 are presented the time space diagrams of the Rule$\triangle$10 using different update schemes.

The rule of this class is *Two-Cycle rules* according to the Li-Packard classification and *Class 2* according to the Wolfram classification.

Formally, the behavior of the rule with Synchronous, Cyclic and Equal Frequency Clocked schemes can be described as

$$\forall\, q_1 \in \mathcal{S}^n, t > t_t, q_t = \begin{cases} q_{t_{t+1}} & \text{if } t - t_t \text{ is odd} \\ q_{t_{t+2}} & \text{if } t - t_t \text{ is even} \end{cases} \tag{4.44}$$

,where t_t is the length of the transient phase p_1.

The rule has a different behavior with the Random update schemes. As shown in Figure 4.27, the spaces is divided into homogeneous region (all zero or all one). Updating the automaton with the Equal Frequency Clocked scheme, that regions remain "fixed": they shift left and right at each time step, but they returns to the same positions after two cycles. Using the Random update schemes, such regions grow, merge and collapse. Regions transformations are determined by edge movement. Transformations never occur in the middle of a region, i.e. a region cannot split into two regions. If a region grows enough to occupied all the spaces, that regions remains constant in the time (a fixed point is reached). The two fixed point are the configurations $\overline{0}$ and $\overline{1}$.

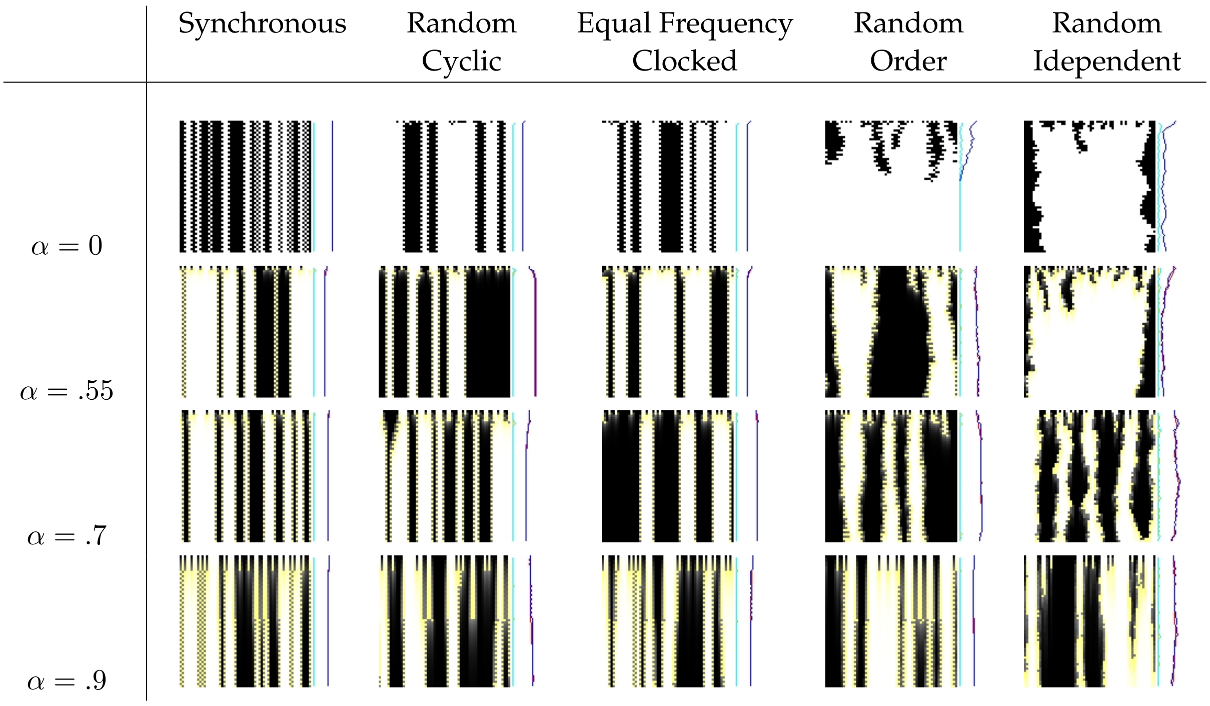


Figure 4.26: *Time space diagrams of Rule* $10\triangle$ *using different update schemes and memory factors.*

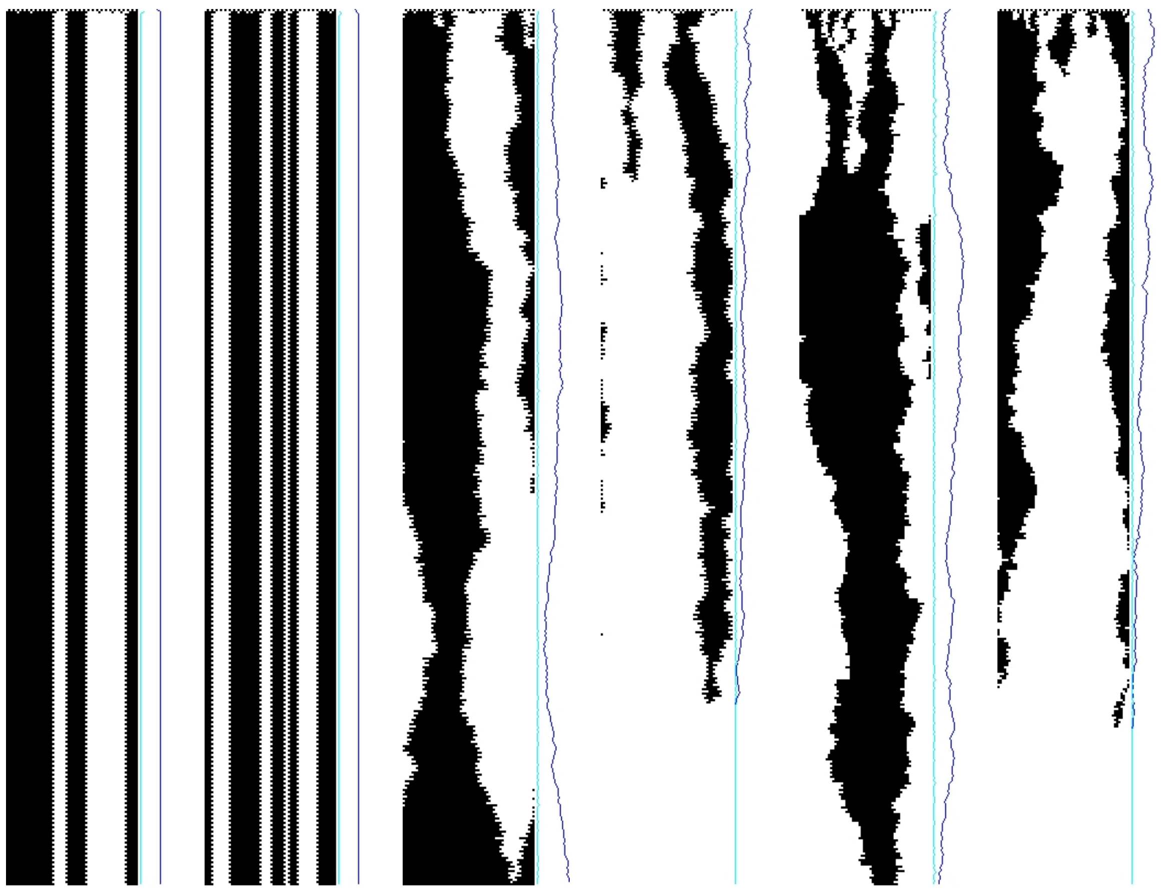


Figure 4.27: *Time space diagrams of Rule* $10\triangle$ *using different update schemes starting from the same initial configuration* ($q_1 = \overline{01}$).

4.3.10 Class $12\triangle$N

The rules Rule $12\triangle$ is a Neighbor-Independent rule that simply maintain any initial configuration over time (i.e. the limiting configuration is equal to the initial configuration. Formally,

$$\forall\, q_1 \in \mathcal{S}^n, t > 1, q_t = q_1 \tag{4.45}$$

According to the Li-Packard classification, this rules is classified as *Fixed point rule* and as *Class 1* according to Wolfram. As shown in Figure 4.28, this rule is the most "stable" of all the rules: the dynamic behavior is not influenced by the update strategy and by the memory factor.

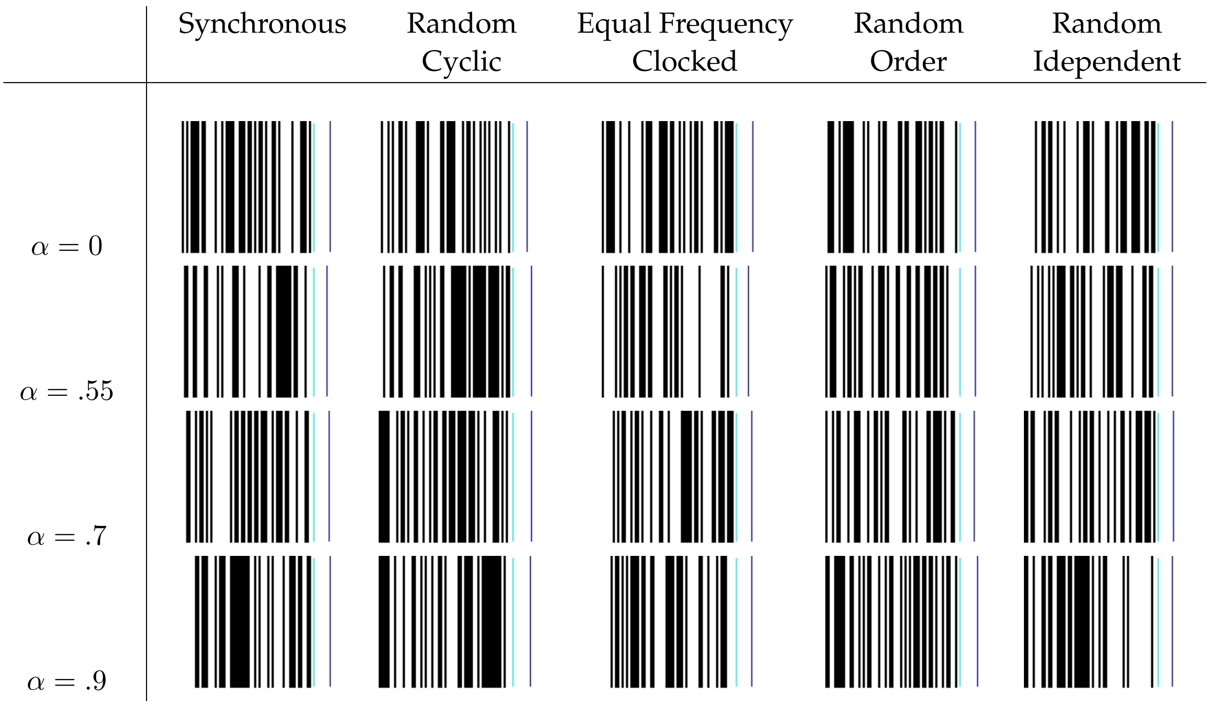


Figure 4.28: *Time space diagrams of Rule* $12\triangle$ *using different update schemes and memory factors.*

4.3.11 Synthesis of the Effects of Asynchrony on 1nCA

In this section we present a synthesis of the effect induced by the different update schemes on the 1nCA. We can classify the 1nCA according to how much the dynamic evolution is influence by the update scheme.

Comparing the dynamic behavior of each automata with the synchronous update scheme with the same automata with a different update scheme, we defined the following 4 classes of asynchrony influence:

- *AS1* – Not influenced by the update schemes, the dynamic behavior does not change varying the update scheme.
- *AS2* – The Random Independent update scheme perturbs the dynamic behavior.
- *AS3* – The Random Independent and Random Order update schemes perturb the dynamic behavior.
- *AS4* – Any update schemes different from synchronous perturbs the dynamic behavior.

In Table 4.8 is shown the Classification summary.

Table 4.8: *Classification summary*

Rule	Bin	Wolfram Class	Li-Packard Class	Class	λ	$R\rho$	μ	Asynchrony Sensitive
$0\triangle$	0000	W1	Null	$0\triangle$TNS	0	0	0	AS1
$1\triangle$	0001	W2	Two-Cycle	$1\triangle$T	0.25	0.375	0.5	AS3
$2\triangle$	0010	W2	Two-Cycle	$2\triangle$	0.25	0.25	0.5	AS4
$3\triangle$	0011	W2	Two-Cycle	$3\triangle$N	0.5	0.5	0.5	AS2
$4\triangle$	0100	W1	Fixed-Point	$4\triangle$	0.25	0.25	0.5	AS1
$5\triangle$	0101	W2	Two-Cycle	$5\triangle$S	0.5	0.5	0.5	AS3
$6\triangle$	0110	W3	Chaotic	$6\triangle$T	0.5	0.5	1	AS4
$7\triangle$	0111	W2	Two-Cycle	$1\triangle$T	0.75	0.625	0.5	AS3
$8\triangle$	1000	W1	Null	$8\triangle$T	0.25	0	0.5	AS1
$9\triangle$	1001	W3	Chaotic	$6\triangle$T	0.5	0.5	1	AS4
$10\triangle$	1010	W2	Two-Cycle	$10\triangle$S	0.5	0.5	0.5	AS3
$11\triangle$	1011	W2	Two-Cycle	$2\triangle$	0.75	0.75	0.5	AS4
$12\triangle$	1100	W2	Fixed-Point	$12\triangle$N	0.5	0.5	0.5	AS1
$13\triangle$	1101	W2	Fixed-Point	$4\triangle$	0.75	0.75	0.5	AS1
$14\triangle$	1110	W1	Null	$8\triangle$T	0.75	1	0.5	AS1
$15\triangle$	1111	W1	Null	$0\triangle$TNS	1	1	0	AS1

We can't solve problems by using the same kind of thinking we used when we created them.

Albert Einstein

5 MDCA - Multilayered Dissipative Cellular Automata

5.1 MDCA Model

In this chapter we proposed a new cellular automata approaches to model, simulate, and control Adaptive Lighting systems. The proposed model, called Multilayered Dissipative Cellular Automata (MDCA), is an Asynchronous, Heterogeneous, Multilayered, and Dissipative Cellular Automata.

MDCA extends the "classic" cellular automata in several ways. The main characteristics of the models are:

- *Asynchrony* - The cells can be updated according to several update schemes, both synchronous and asynchronous. This characteristic is required because the model is suitable for distributed Adaptive Lighting installation, where there is no evidence of a global clock.
- *Heterogeneity* - Cells are heterogeneous, in terms of space of the states and transition rule.
- *Multilayered* - The cellular space is a hierarchical structure, deriving from the structure of the Multilayered Automata Networks (Bandini and Mauri, 1999). A schematization of the MDCA hierarchical structure is shown in Figure 5.1.
- *Openess* - MDCA is a Dissipative Cellular Automata (Roli and Zambonelli, 2002) (Zambonelli et al., 2002), i.e the dynamic behavior of the automata is influenced by the external environment and influences the external environment. This characteristic is fundamental for the Adaptive Lighting system, that are open system comprising sensors and actuators.

Such features are useful for designing systems composed of several distributed interacting components. MDCA can be used both to simulate the behavior of such distributed systems and to control the real installations. Moreover it can be used in centralized situations to control peripheral components. In the following sections we introduce the main elements of the model.

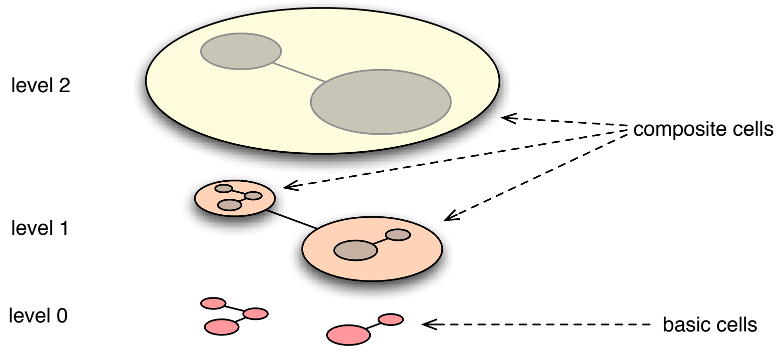


Figure 5.1: *Schematization of the MDCA multilayered structure.*

5.1.1 Basic Cell

As a cellular system, the fundamental element of MDCA is the *cell*. As a cellular system, the fundamental element of MDCA is the *cell*. There are two kind of cells: *basic cells* and *composed cells*.

The *basic cells* are basic building block of the MDCA. Elaborate behaviors are defined composing such basic cells. Each basic cell c, schematized in Figure 5.2, is a triple $(\mathcal{R}_c, \mathcal{E}_c, f_c)$ where

- $\mathcal{R}_c = \{\mathcal{S}_{r_{c,1}}, \mathcal{S}_{r_{c,2}}, \cdots, \mathcal{S}_{r_{c,3}}\}$ is a finite set of "organs of the cell", called *receptors*, able to respond to external stimulus. Each receptor is characterized by a set of state that can assume $\mathcal{S}_{r_{c,n}}$. The receptor can be "connected" to a *transmitter* of an other cell, or to a *receptors* of the its father cell. When connected, it assume the value of the other cells external state. A receptor not connected to any other cells assumes the *null* state.

- $\mathcal{E}_c = \{\mathcal{S}_{e_{c,1}}, \mathcal{S}_{e_{c,2}}, \cdots, \mathcal{S}_{e_{c,3}}\}$ is a finite set of states "exposed" to the other cells (*external states*);

- $f_c : \mathcal{R}_c \to \mathcal{E}_c$ is the *transition rule*, determining the new value of the external states.

The receptors and externals set of states are nonempty, finite and ordered set of state values. The following "primitive" data sets are defined in the model:

- int: signed integer number

- $float$: floating point number

- $boolean = \{\ true,\ false\}$

The dimension (in terms of numbers of bits) of the numeric data types depends on the specific implementations. We introduce other two data sets:

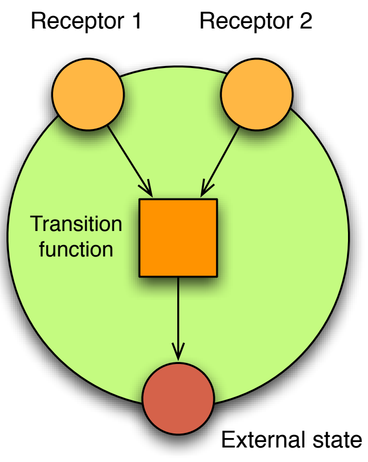


Figure 5.2: *Schematization of a basic cell.*

- $number = int \cup float$: a signed integer number or a floating point number
- $any = int \cup float \cup boolean$: any type of data.

MDCA is an *open system*, in the sense that the dynamic behavior of the automata is influenced by the external environment and influences the external environment. We defined the following "special" basic cells:

- *sensor cell* is a cell that can be forced from the external to change its state;
- *actuator cell* influences the external environment according to its external state;
- *open cell* is both a sensor cell and an actuator cell;

An example of MDCA sensors and actuator cells is shown in Figure 5.3. The external state of the sensor cell shown in the example depends on the value obtained by the ultrasonic distance measurement sensor. In the second example, the activation of the fog machine depends on the status of the actuator cell.

There are several predefined basic cells. The basic cells set is not intended to be "minimal", but to allows the Adaptive Lighting developer to easy compose the desired behavior. Such cells fall into a number of groups:

- Arithmetic Integer (Table 5.1) - This cells performs arithmetic operations on integer numbers.
- Floating Point (Table 5.2) - This cells performs operations on floating point numbers. This cells are available only for the MDCA Virtual Machines supporting the floating point numbers.
- Logic (Table 5.3) - This cells performs bitwise operations, i.e. they operates on one or two bit patterns at the level of their individual bits.

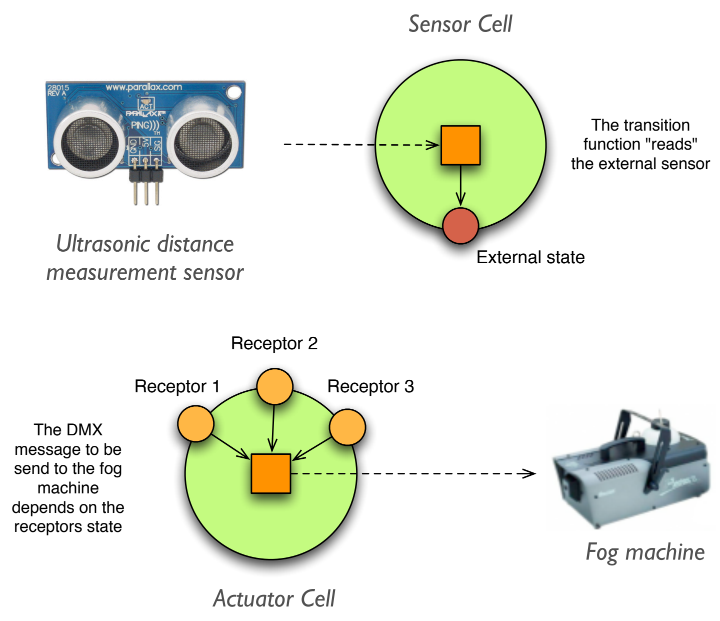

Sensor Cell
The transition function "reads" the external sensor
External state
Ultrasonic distance measurement sensor
Receptor 2
Receptor 1
Receptor 3
The DMX message to be send to the fog machine depends on the receptors state
Fog machine
Actuator Cell

Figure 5.3: *Schematization of an MDCA sensors and actuator cells.*

- Miscellaneous (Table 5.4) - Miscellaneous cells, such as *inv* cell, that copy the receptor state to external state, *if*, *equal*, and *notEqual* cells.

The set of basic cells can be extended according to requirements of the specific applications.

5.1.2 Composite Cell

The composite cells are automata composed of cells, both basic and composed. Each composite cell c is is a triple $(\mathcal{R}_c, \mathcal{E}_c, f_c)$ where

- $\mathcal{R}_c = \{\mathcal{S}_{r_{c,1}}, \mathcal{S}_{r_{c,2}}, \cdots, \mathcal{S}_{r_{c,3}}\}$ is a finite set of *receptors* of the cell able to respond to external stimulus. Each receptor is characterized by a set of state that can assume $\mathcal{S}_{r_{c,n}}$. The receptor can be "connected" to other cells *Transmitter*. When connected, it assume the value of the other cells external state. Each receptor can be connect the receptor of an internal cell.

- $\mathcal{E}_c = \{\mathcal{S}_{e_{c,1}}, \mathcal{S}_{e_{c,2}}, \cdots, \mathcal{S}_{e_{c,3}}\}$ is a finite set of *external states*. The external states must be connected to the external states of of an internal cell and they assume the values of the external state they are connected to.

- $\mathcal{C}_c = \{c_{c,1}, c_{c,2}, \cdots, c_{c,n}\}$ is a finite set of internal cells (*sub–cells*).

- $\mathcal{V}_c \subset \mathcal{C}_c \cup \mathcal{R}_c \times \mathcal{C}_c \cup \mathcal{E}_c$ is a set of directed arcs, connecting the receptors with the sub–cells, the sub–cells together, and the sub-cells to the external states.

Table 5.1: *Arithmetic Integer Basic Cells*

Name	Receptor	Transition rule	External State
add	r1:int, r2:int	$e1 = r1 + r2$	e1:int
sub	r1:int, r2:int	$e1 = r1 - r2$	e1:int
mul	r1:int, r2:int	$e1 = r1 * r2$	e1:int
div	r1:int, r2:int	$e1 = r1/r2$	e1:int
rem	r1:int, r2:int	$e1 =$ remainder of division of $r1$ by $r2$	e1:int
less	r1:int, r2:int	$e1 = r1 < r2$	e1:boolean
lessEq	r1:int, r2:int	$e1 = r1 \leq r2$	e1:boolean
great	r1:int, r2:int	$e1 = r1 > r2$	e1:boolean
greatEq	r1:int, r2:int	$e1 = r1 \geq r2$	e1:boolean

Table 5.2: *Floating Point Basic Cells*

Name	Receptor	Transition rule	External State
addF	r1:float, r2:float	$e1 = r1 + r2$	e1:float
subF	r1:float, r2:float	$e1 = r1 - r2$	e1:float
mulF	r1:float, r2:float	$e1 = r1 * r2$	e1:float
divF	r1:float, r2:float	$e1 = r1/r2$	e1:float
lessF	r1:float, r2:float	$e1 = r1 < r2$	e1:boolean
lessEqF	r1:float, r2:float	$e1 = r1 \leq r2$	e1:boolean
greatF	r1:float, r2:float	$e1 = r1 > r2$	e1:boolean
greatEqF	r1:float, r2:float	$e1 = r1 \geq r2$	e1:boolean
intToFloat	r1:int	$e1 =$ convert int $r1$ to float	e1:float
floatToInt	r1:float	$e1 =$ convert float $r1$ to int	e1:int

Table 5.3: *Logic Basic Cells*

Name	Receptor	Transition rule	External State
and	r1:any, r2:any	$e1 = r1$ bitwise and $r2$	e1:any
or	r1:any, r2:any	$e1 = r1$ bitwise or $r2$	e1:any
xor	r1:any, r2:any	$e1 = r1$ bitwise xor $r2$	e1:any
not	r1:any, r2:any	$e1 = r1$ bitwise not $r2$	e1:any

Table 5.4: *Miscellaneous Basic Cells*

Name	Receptor	Transition rule	External State
inv	r1:any	$r1$	e1:boolean
equal	r1:any, r2:any	$r1 = r2$	e1:boolean
notEqual	r1:any, r2:any	$r1 \neq r2$	e1:boolean
if	r1:bool, r2:any, r3:any	$e1 = \begin{cases} r_2 & \text{if } r1 = true \\ r_3 & \text{if } r1 = false \end{cases}$	e1: any
shiftLeft	r1:any, r2:int	$r1$ left shifted by $r2$ bits	e1:any
shiftRight	r1:any, r2:int	$r1$ right shifted by $r2$ bits	e1:any

A different update scheme can be associated to each cell. For example, the lower layer composite cell (composed only of basic cells) can adopt a synchronous scheme.

An example of composed cell is shown in Figure 5.4. In the example, we implemented the the 1NCA Rule 6△ as MDCA composed cell. The Rule 6△ behavior can be summarized as: computer the *xor* between the internal state and, alternatively, the state of the left and right neighbors. The automaton is characterized by:

- *Receptors* - $\mathcal{R}_c = \{\mathcal{S}_{r_{c,1}}, \mathcal{S}_{r_{c,2}}\}$, where $r_1 \in boolean, r_2 \in boolean$ connected to the external state of the left and right cells.
- *External states* - $\mathcal{E}_c = \{\mathcal{S}_{e_{c,1}}, \mathcal{S}_{e_{c,2}}\}$, where $e_1 \in boolean$.
- *Sub–cells* - $\mathcal{C}_c = \{c_1, c_2, c_3, c_4, c_5\}$, the set of internal cells. The initial external state

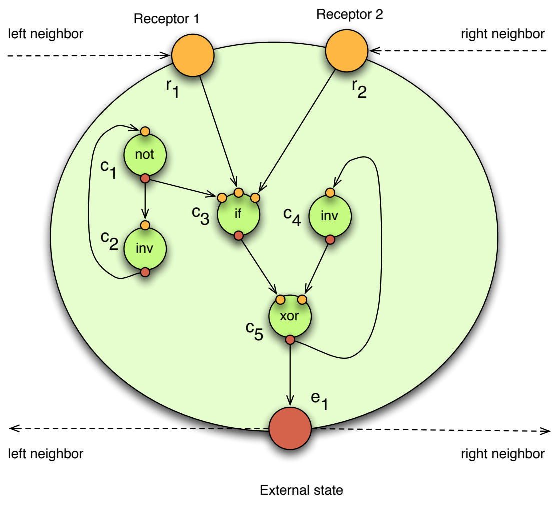


Figure 5.4: *Example of implementation of the 1NCA Rule 6△ as MDCA composed cell.*

of all the cells is $False$.

- *Connections* - the set of directed arcs, as shown in the figure.

In two cycles this automaton computes as a Rule 6△ cell, as shown in Table 5.5

Table 5.5: *Dynamic evolution of the automata presented in the example.*

t	r_1	r_2	c_1	c_2	c_3	c_4	c_5	e_1
0	True	False	False	False	False	False	False	False
1	True	False	True	False	True	False	False	False
2	True	False	True	True	True	False	True	False
3	True	False	False	True	True	True	False	True
4	True	False	False	False	False	False	False	False

Considering the Elementary Cellular Automata, introduced by Wolfram (Wolfram, 1982). As shown in Figure 5.5, is very easy to create an MDCA cell that simulate the behavior of any of the Elementary Cellular Automata.

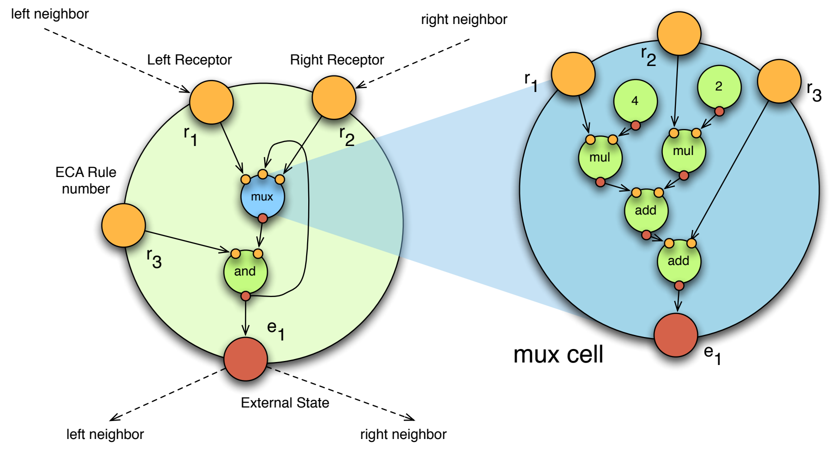


Figure 5.5: *Implementation of a parametric Elementary Cellular Automata with MDCA.*

5.1.3 Update Schemes

As shown in Chapter 4, the update scheme has a great influence the system behavior. A "flexible" update scheme is key feature of MDCA. The proposed model supports several update schemes (e.g. synchronous, cyclic, clocked) by defining two parameters.

The update scheme is determined by the parameters d_i, p_i associated to each cell i. The parameter d_i determines the delay (in terms of time step) before the first update.

The parameter p_i, determines the period of the update of the cell i, i.e. how many time steps the cell i will wait in order to be updated. Both the parameters are relative to the container cell. Considering the example depicted in Figure 5.6. The cells c_1, c_2, c_3 are subcells of the cell c_4, the cells c_5, c_6, c_7 are subcells of the cell c_8. The update periods are the following:

$$d_1 = 1, \;\; d_2 = 2, \;\; d_3 = .5, \;\; d_4 = 1, \;\; d_5 = 1, \;\; d_6 = 2, \;\; d_7 = .5, \;\; d_8 = 2$$

According to such update scheme, the cells c_1, c_2, c_3 update twice as fast as the cells c_5, c_6, c_7, because the cell c_4 update periods is half than the cell c_8.

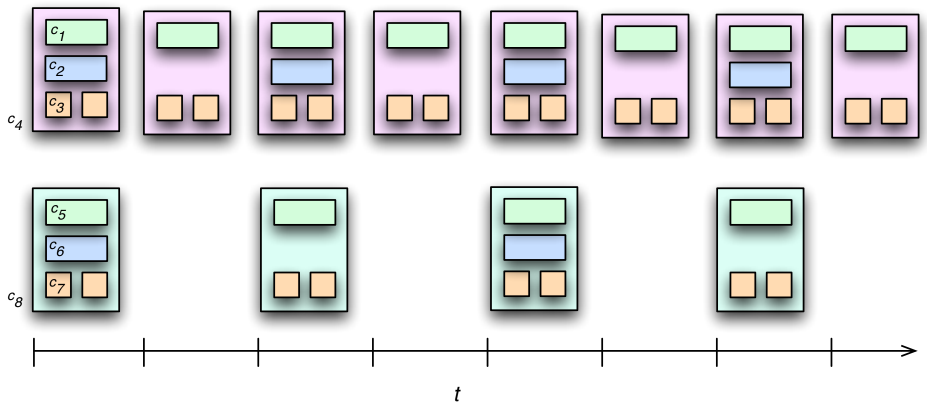


Figure 5.6: *Example of cells with different update periods.*

As shown in Figure 5.7, varying the two parameters is it possible to obtain the behavior of the *Synchronous*, *Cyclic*, *Clocked*, and *Clocked* update schemes. The default values for the parameter are $d_i = 0$ and $p_i = 1$. This setup corresponds to the *Synchronous* update scheme: all the cells are updated in parallel at each time step. The *Cyclic* update schemes are obtained associating to each of the n cells a different value of d_i between 0 and $n-1$, and the same period $p_i = n$. The *Clocked* update schemes are characterized by different values of d_i and p_i. The *Equal Frequency Clocked* is characterized by different values of d_i but the same value of p_i for every cells of the automaton.

5.2 MDCA Programming

In this section we describe the MDCA programming language and tools. The first part of this section introduce the MDCA language, a textual cellular automata programming language. The language allows the user to create automata creating new cells and combining existing cells. In the successive section we present the MDCA visual programming and the visual editor.

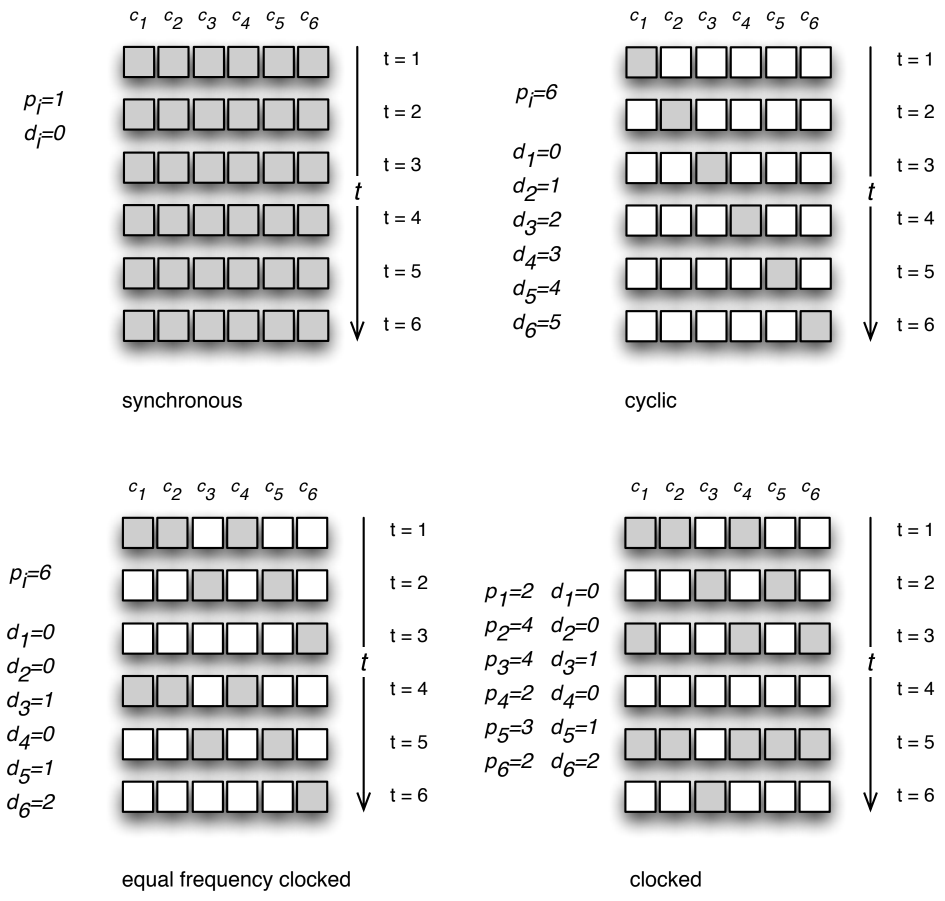


Figure 5.7: *An example of different update scheme obtained varying the d_i and p_i parameters.*

5.2.1 MDCA Language

The MDCA Language is a cellular automata programming language. It allows the user to create automata creating new cells and combining existing cells.

A MDCA programs consist in a set of cells descriptions. A cell description determines the *receptors*, *sub–cells*, and *external states* composing the cell, the wiring and update schemes. The order of the elements inside a cell definition is not relevance. Nevertheless the order of the cells definitions is relevance: all the subcells of a cell must be defined before the cell itself. An example of cell definition is the following:

```
cell rule6 {
    // Receptors and external state
    receptor boolean r1;
    receptor boolean r2;
    external boolean e1;

    // Cells
    cell not c1;

    cell inv c2;
    cell if c3;
    cell inv c4;
    cell xor c5;

    // Initial values
    c1.e1 = false;
    c2.e1 = false;
    c3.e1 = false;
    c4.e1 = false;
    c5.e1 = false;

    // Wiring
    c1.e1 -> c2.r1;
    c2.e1 -> c1.r1;
    c1.e1 -> c3.r1;
    r1 -> c3.r2;
    r2 -> c3.r3;
    c3.e1 -> c5.r1;
    c4.e1 -> c5.r2;
    c5.e1 -> c4.r1;
    c5.e1 -> e1;
}
```

The MDCA syntax is influenced by the Java syntax. MDCA source code is *free–form* which allows arbitrary use of whitespace and ends of lines to format code. Comments may appear either between the delimiters /* and */, or following // until the end of the line.

A MDCA programs consist in a set of cells definitions. The order of the definition is not relevant. Each cell definitions begins with the *cell* keyword followed by the cells name (*identifier*) and the cell definition.

Every identifier is made from the following characters, starting with a letter:

- Letters: a–z, A–Z
- Digits: 0–9
- Underscore: _

No identifier can be the same as a MDCA keyword or pre–defined cell name. The MDCA keywords are the following: *cell*, *receptor*, *external*, *int*, *float*, *boolean*, *true*, *false*, *null*.

Cells definitions are enclosed in braces ({ and }) to limit their scopes. A cell definitions consists in a list of the following elements:

- *Receptor Declaration* Declaration of a cell receptor, expressed with the following syntax:

  ```
  receptor <receptor type> <receptor identifier>;
  ```

  where the *receptor type* is on of the built–in data type and *receptor identifier* is an identifier. The keywords *int*, *float*, and *boolean* specify built–in data types.

- *External State Declaration* Declaration of a cell external state, expressed with the following syntax:

  ```
  external <external state type> <external state identifier>;
  ```

  where the *external state type* is on of the built–in data type and *external state identifier* is an identifier.

- *Sub–Cell Declaration* Declaration of a sub–cell, expressed with the following syntax:

  ```
  cell <cell type> <cell identifier>;
  ```

  where the *cell type* is an identifier of a pre–defined or a user–defined cell type and *cell identifier* is an identifier for the sub–cell. The sub–cells scope is limited to the cell definition.

- *Initial Value Assignment* Assignment of the initial external state of a sub–cell, expressed with the following syntax:

  ```
  <sub-cell identifier>.<sub-cell external state identifier> = <literal>;
  ```

  where *sub–cell identifier* is an identifier of a sub–cell, *sub-cell external state identifier* is an identifier of an external state of the sub–cell, and *literal* is the value to be assigned to the external state of the cell. A *literal* is the source code representation of a value of a built–in data type. The literals are the *numberic literals*, floating–point numbers expressed according to the Java syntax, *boolean literal*, *true* and *false*, and the *null literal*, represented by the keyword *null*.

- *Initial Delay Assignment* Assignment of the initial delay before the first update of a sub–cell, expressed with the following syntax:

  ```
  <sub-cell identifier>.delay  = <number>;
  ```

  where *sub–cell identifier* is an identifier of a sub–cell and *number* is the initial delay in terms of time step.

- *Period Assignment* Assignment of the update period of a sub–cell, expressed with the following syntax:

  ```
  <sub-cell identifier>.period  = <number>;
  ```

  where *sub–cell identifier* is an identifier of a sub–cell and *number* is the length of the period expressed in time step.

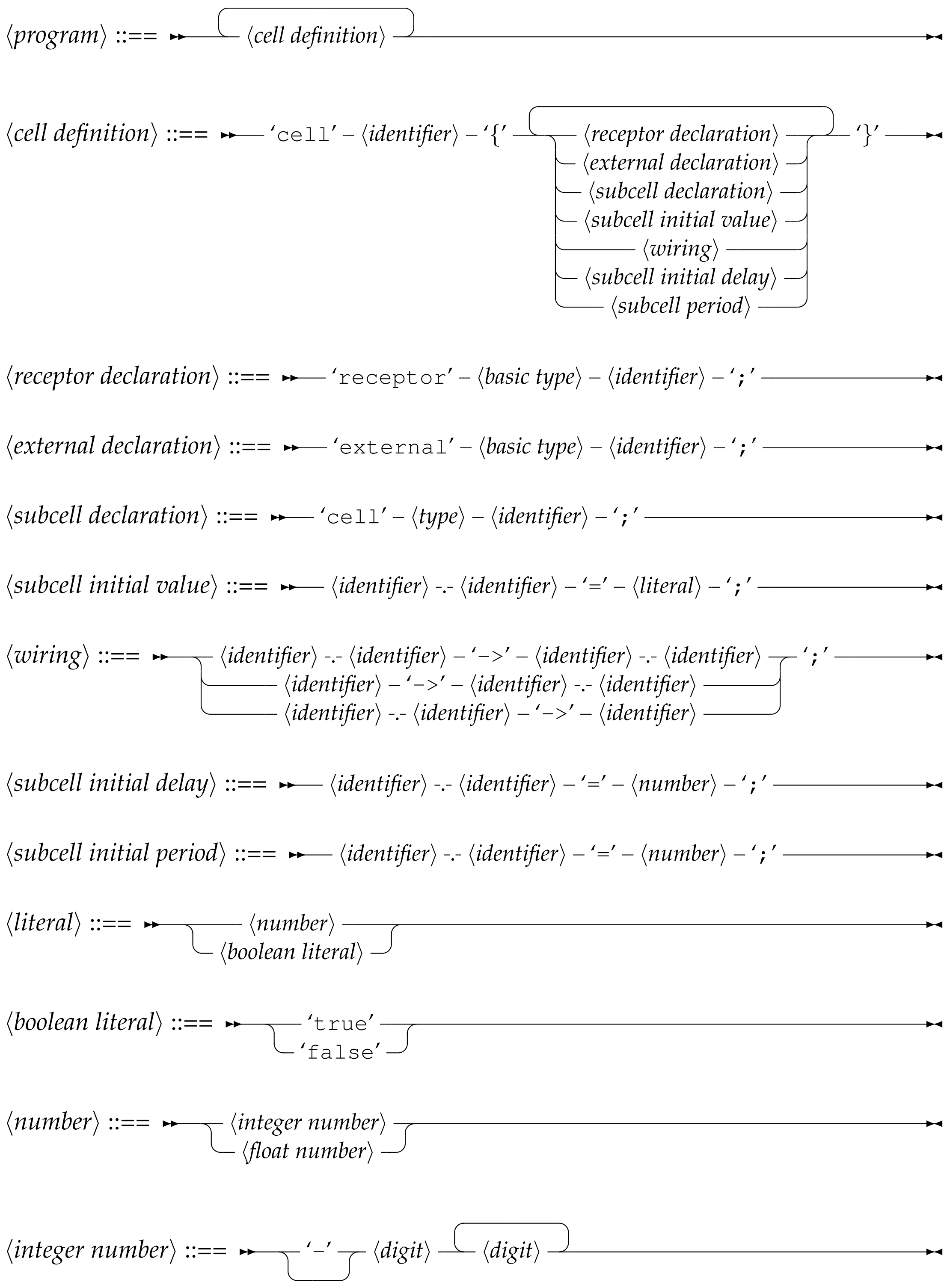
- *Wiring*: Definition of an arc between two cells, or a receptor and a cell or a cell an an external state. Direct arc between a receptor and an external state are not allowed.

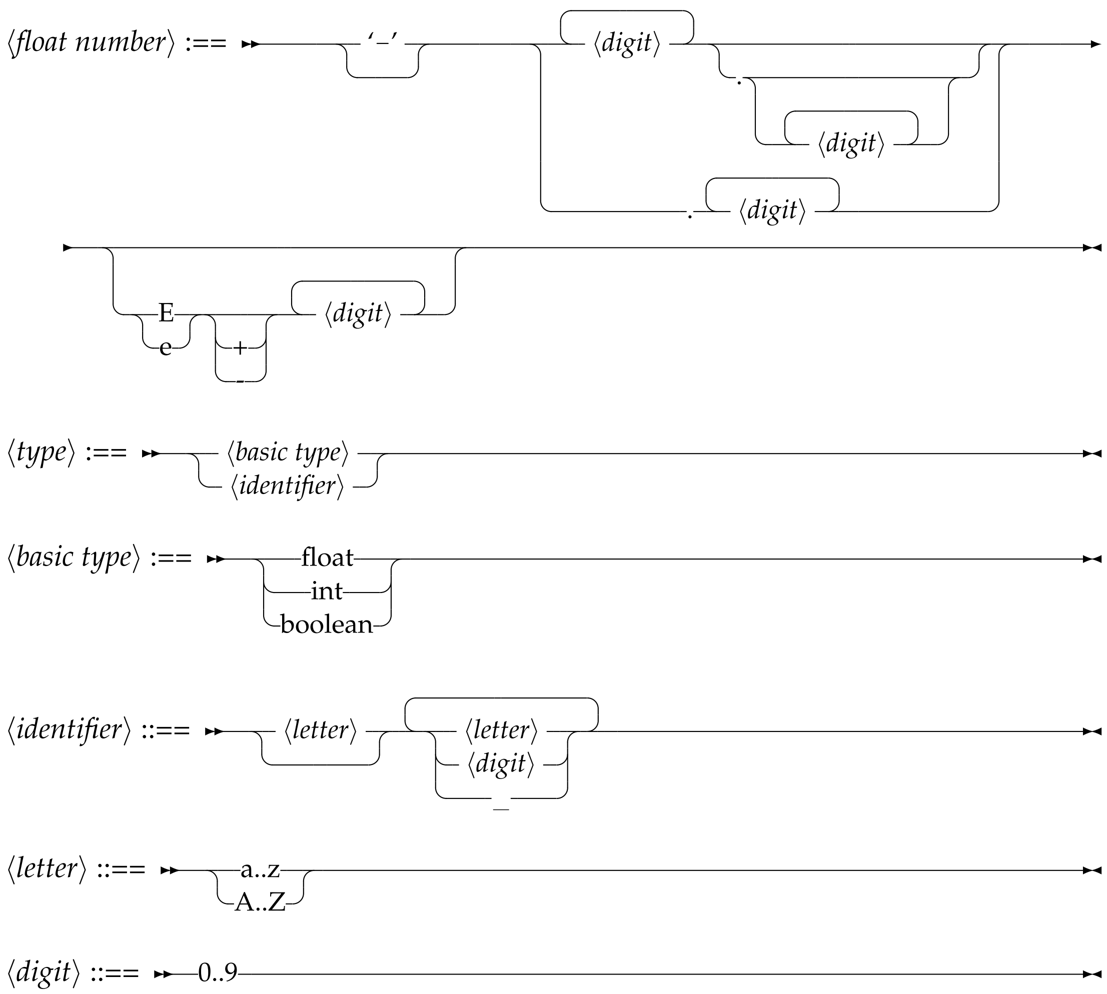
  ```
  <source> -> <target>;
  ```

  where *source* can be a receptor identifier or a sub–cell receptor identifier and the *target* can be an external state identifier of a sub–cell external state identifier. The sub–cell receptor and external state identifier are expressed with the following syntax:

  ```
  <sub-cell identifier>.<receptor/external state identifier>
  ```

The syntax diagram for the MDCA language is the following:


⟨program⟩ ::== ⟨cell definition⟩
⟨cell definition⟩ ::== 'cell' – ⟨identifier⟩ – '{' ⟨receptor declaration⟩ ⟨external declaration⟩ ⟨subcell declaration⟩ ⟨subcell initial value⟩ ⟨wiring⟩ ⟨subcell initial delay⟩ ⟨subcell period⟩ '}'
⟨receptor declaration⟩ ::== 'receptor' – ⟨basic type⟩ – ⟨identifier⟩ – ';'
⟨external declaration⟩ ::== 'external' – ⟨basic type⟩ – ⟨identifier⟩ – ';'
⟨subcell declaration⟩ ::== 'cell' – ⟨type⟩ – ⟨identifier⟩ – ';'
⟨subcell initial value⟩ ::== ⟨identifier⟩ -.- ⟨identifier⟩ – '=' – ⟨literal⟩ – ';'
⟨wiring⟩ ::== ⟨identifier⟩ -.- ⟨identifier⟩ – '->' – ⟨identifier⟩ -.- ⟨identifier⟩ ⟨identifier⟩ – '->' – ⟨identifier⟩ -.- ⟨identifier⟩ ⟨identifier⟩ -.- ⟨identifier⟩ – '->' – ⟨identifier⟩ ';'
⟨subcell initial delay⟩ ::== ⟨identifier⟩ -.- ⟨identifier⟩ – '=' – ⟨number⟩ – ';'
⟨subcell initial period⟩ ::== ⟨identifier⟩ -.- ⟨identifier⟩ – '=' – ⟨number⟩ – ';'
⟨literal⟩ ::== ⟨number⟩ ⟨boolean literal⟩
⟨boolean literal⟩ ::== 'true' 'false'
⟨number⟩ ::== ⟨integer number⟩ ⟨float number⟩
⟨integer number⟩ ::== '-' ⟨digit⟩ ⟨digit⟩



5.2.2 MDCA Visual Programming

Visual Programming refers to any system that allows the user to specify a program in a two or more dimensional fashion (Myers, 1990). Visual programming languages have a long history during which there have been many different languages developed with the common goal of ameliorating the difficulties of programming (Edmonds et al., 2005). There is no scientific evidence that visual programming is generally better or easier than text (Goodell et al., 1999). However visual programming languages, such Max, Pure Data, vvvv are more popular that text languages in designer and artists communities.

MDCA Visual Programming Editor is a graphical programming environment. Using the MDCA Visual Programming Editor (MDCA VPE), the MDCA textural language is hidden to the user.

The user interface of MDCA VPE has one main window and any number of sub–windows. Each window is a cell editor. Programming through MDCA VPE works by connecting cells. The editor allows to add, remove and edits the sub–cells, the receptors, and the external states, and define the wiring.

As shown in Figure 5.8 the user interface is inspired both by Pure Data and Quartz Composer.

There are several types of elements in the user interface: cell, receptor, external, value, and comment.

Cells are the basic elements of the visual language, similar to routines in traditional programming languages. A cell is represented as a rectangle, with the type of the cell written as title. The text the user types into the title of a cell box determines how many and what kinds of receptor and externals the box will have. Circles on the cell represent connections, with receptors on the left side of a cell and externals on the right side. Each receptors and externals are characterized by a name and a type.

Receptor and external as created writing the words "receptor" or "external" in the box title. The receptor boxes are characterized by a single receptor with a name and a

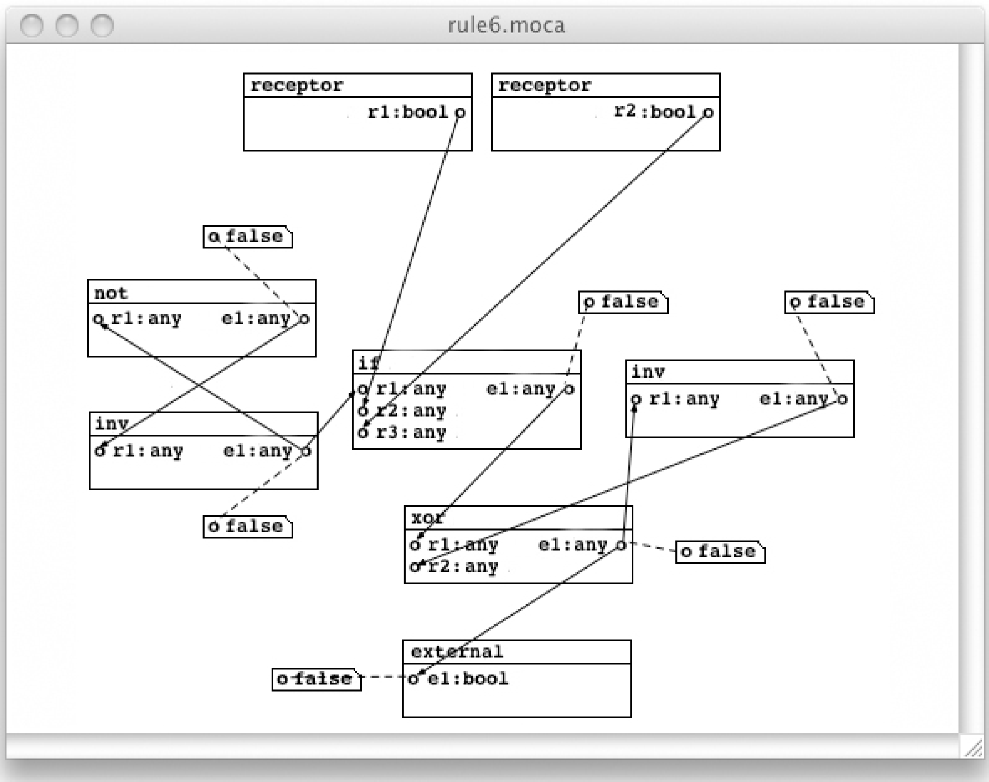


Figure 5.8: *A screenshot of the MDCA Visual Programming Editor editing the 1NCA Rule 6△ cell.*

type, and external boxes by a single external state.

Value boxes are boxes with a single numeric or boolean value written inside. Such boxes are used as initial values. They can be connected only to the externals connections.

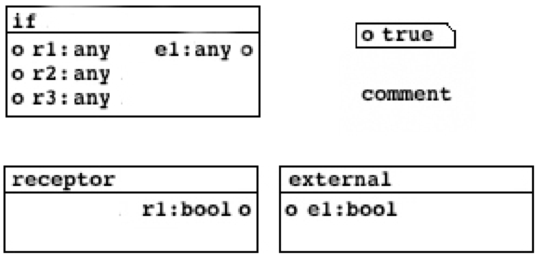


Figure 5.9: *Example of different types of boxes: a cell of type* if*, a value* true*, a comment, a receptor, and an external state. Circles on the cell represent connections, with receptors on the left side of a cell and externals on the right side.*

5.3 MDCA Environment

In this section we introduce the MDCA Embedded Environment (MDCAee). It is a run-time environment for the MDCA cells suitable for the class of microcontrollers typically used in distributed control systems. At a different level of granularities, the MDCAee node itself can be viewed as a cell of large system, composed of several MDCAee nodes. In order to act as an environment for the cells, MDCAee offers some functionalities of a "traditional" operating system, such as hardware abstraction, scheduling, memory management, I/O. MDCAee is a simple operating system designed to run on 8-bit and 16-bit microcontroller. Moreover, it can run as a user process on a standard desktop operating system. This feature simplified the development (and the debug) of MDCA cells. The operating system is written in ANSI C language, together with a number of short sections of device-dependent code written in the assembly.

As depicted in Figure 5.10, MDCAee is composed of two layer: the *kernel* and the *cellular space*. The aim of the kernel is to provide the basic functionalities such as hardware abstraction, scheduling, memory management, I/O to the upper layer. The cellular space contains all the cells. In the following sections, we introduce the main components of MDCAee.

5.3.1 Kernel

The MDCAee Kernel is composed of the MDCAe Core and the devices drivers. The MDCAee Core contains the main modules of the operating system, those are in charge

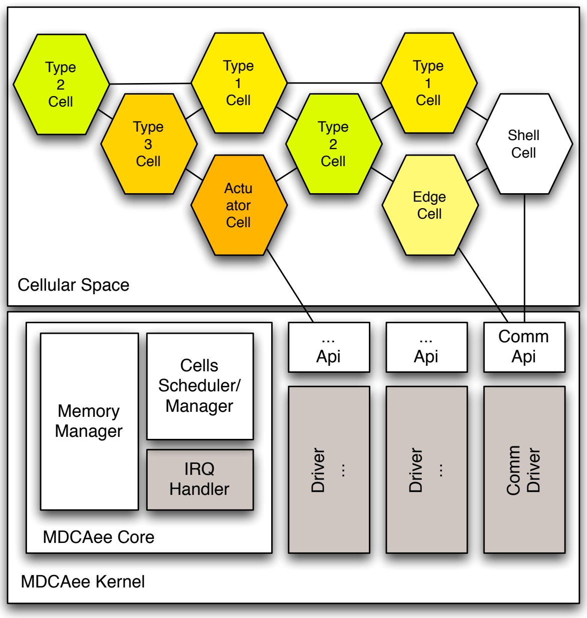


Figure 5.10: *Schematization of the main components of MDCAe. The gray rectangles represent the device-dependents modules.*

to share the resources (i.e. the memory and the CPU time) between the cells. The MDCAee Core comprised:

- Cells Scheduler/Manager, that relays on the interrupt handler. The aim of this component is manage the cells and invoke the cells update functions.
- Interrupt Handlers are hardware–dependent callback routine whose execution is triggered by the reception of an interrupt. Interrupt are used both by the device driver and the scheduler.
- Memory Manager, a very simple heap–based, memory allocator. Heap memory is an internal memory pool that cells use to dynamically allocate memory as needed.

The Cells Scheduler/Manager is the most interesting component of the MDCAee kernel. It is quite different from the scheduler of both the embedded and the desktop operating systems because MDCAee is ground on *cells* instead of *processes*.

The scheduler, invoked by an interrupt hander, cycles on the cells list and execute the update function for all the cells that need to be updated and are not in *inactive* state. To each cell is associated an update frequency. For each cells, if the time passed since the last update is grater than the waiting period, the update function is executed. For the composed cell, the subcells are updated according to their *initial delay* and *period* parameters, as described in Chapter 5. As shown in Table 5.6, there are 7 update frequencies; to each update frequencies is associated a waiting period.

Frequencies	Very Low	Low	Medium	High	Very High
Waiting period	1 s	100 ms	20 ms	5 ms	1 ms

Table 5.6: *Cell update frequencies and associated waiting period.*

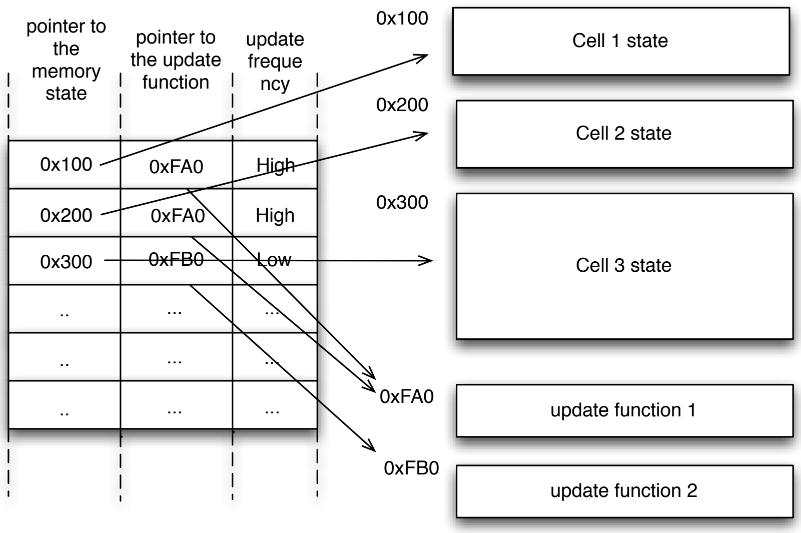


Figure 5.11: *The memory structure used by the Cells Scheduler/Manager.*

Interrupt handlers are functions that are activated synchronously with peripheral hardware interrupt sources, CPU exceptions, and software interrupt instructions. Interrupt handlers can be defined for each interrupt source. If an interrupt occurs while a cell update is running, the function is temporarily interrupted and the interrupt handler (corresponding to the interrupt source that occurred) is executed. All of the interrupt handlers run at a higher priority than the cells, therefore cells do not run until the interrupt handler has finished. If multiple interrupt handlers are activated, task execution does not continue until all of the interrupt handlers have finished processing.

To manage the cells waiting period, MDCAee uses a timer interrupt that occurs at intervals of 1 ms. The timer handler raises flags after 1 ms, when the scheduler is executed, it checks these flags to determinate which cells need to be updated.

The memory structure used by the Cells Scheduler/Manager is shown in Figure 5.11. As shown in the figure, to each cell in the list, there are associated a pointer to the update function, a pointer to the memory area containing the cell state and the cell update frequency. The update function also determinate the cell type. The update function is a common C function. When it is executed, the pointer to the cell state is passed as an argument.

There is no *process identifier* to uniquely identify a cell. In fact, the Cells Scheduler/Manager does not provide any function to manage individual cells according to an id. This is a precise choice in the design of the operating system. The choice is motivated by the cellular environment metaphor. A cell can destroy itself or its neighborhood, but cannot have direct effect on distance cells. Moreover, the Cells Scheduler/Manager provides basic functionalities for executing a function on all the cells of a particular kind. This function is intended for development, debug, special modes or emergency response. For example, it can be use for disable all the actuators cell in response to a not–expected condition, or to disable all the communications cells during a node setup. The cells can be frozen. If a cell is frozen, its update function will non be executed.

The memory manager is in charge to dynamically allocate the memory for the cells, scheduler and drivers. Dynamic memory allocation is important aspect of an operating system. An efficient dynamic memory allocator improves the performance of operating systems.

MDCAee nodes can have very different memory and timing requirements to another. A single memory allocation algorithm will only ever be appropriate for a subset of hardware and applications. In this section we presents three simply memory allocation strategies implemented in the MDCAee.

The operating system can be extended with more advanced memory allocator schemes designed for embedded devices, such as the algorithm presented in (Min et al., 2007). A quite old but useful survey on dynamic memory allocation can be found in (Wilson et al., 1995).

- *Simple Memory Allocation* is the simplest scheme of all. It does not permit memory to be freed once it has been allocated. Despite this, it is suitable for a large number of applications. The algorithm simply subdivides a the heap area into smaller blocks as requests for memory are made. This scheme can be used if the application never deletes any cells and the memory required by each cell is constant after the initial allocation.

- *First Fit Memory Allocation* algorithm basically extends the Simple Memory Allocation, adding a list of free blocks. When the allocator will receive a request of n bytes, it will lookup the list for an available block to fit n bytes. The algorithm select the first block equal or greater than n. This approach will lead to the undesired effect of memory fragmentation (i.e. larger block will be split into smaller ones to fit the requests and may end up with the impossibility of allocating n bytes although the sum of all free, but smaller, blocks is greater than n.

- *Fixed–Size Blocks Memory Allocator* is a simple heap–based memory allocator. Heap memory is an internal memory pool that cells use to dynamically allocate memory as needed. The Memory Manager manages a list of free blocks of memory of the same size. A benefit of the fixed–size block memory pool is that this approach do not need to store allocation metadata for each allocation block. This provides a substantial space savings for small allocation. Allocating and releasing of memory blocks is performed in constant time.

The device drivers act as an abstraction layer between the hardware and the cells. The drivers are composed of a standard interface and an hardware–specific implementation. The driver interfaces is common to different implementation, so the cells code is independent from the underlying hardware (for example, the same cell update function can be compiled on for a microcontroller or a desktop, without modification).

Communications can be used for:

- exchanging state information with the other nodes, mimic the interaction mechanism of the cells;
- acquiring external (not on-board) inputs (e.g. motion sensor, light sensors);
- controlling external devices (e.g. DMX-controlled lights);
- monitoring and control the nodes.

MDCAe does not provide any high level communication facilities. It simply provides functions for sending messages or bytes over communication lines.

There are two kind of communication line defined in MDCAee: *message-oriented* and *char-oriented*. The kind of the communication line does not depends on the communication technology but depends on the communication line aim.

The message-oriented communication lines are primary used to exchange state between the cells. Char-oriented communication line are used to interact with external devices, and for monitor and control the nodes.

In the developed prototype, only serial communication driver is present. The developed drivers supports both message–oriented and char-oriented lines. The char-oriented communication is used to manage and debug the nodes, the message-oriented communication is used to share information between the neighbors.

5.3.2 Cellular Space

The cellular space is the environment in which the MDCA cell exist and their update functions are periodically executed. The main component of the cellular space is the MDCA virtual machine, which executes the cell *bytecode*. The bytecode is the form of cells definitions that the MDCA Virtual Machine understands. Through the *MDCAc* compiler, implemented in the Java language, the MDCA source code is compiled into *MDCA bytecode*.

Consider the following cell definition:

```
cell pow2 {
    receptor float r1;
    external float e1;

    cell mul c1;
    c1.e1 = 0;
    c1.delay = 0;
    c1.period = 1;

    r1 -> c1.r1;
    r1 -> c1.r2;
    c1.e1 -> e1;
}
```

The MDCAc compiler translates the code above into MDCA bytecode as follows:

```
pow2,1,1,null,10,mul,0,1,3,3,0
```

Such bytecode correspond to the following representation:

0	1	2	3	4	5	6	7	8	9	10
name	#rcpts	#exts	r1	e1	c1	c1.delay	c1.period	c1.r1	c1.r2	c1.e1
pow2	1	1	null	10	mul	0	1	3	3	0

At the first position there is the identifier of the cell, that the numbers of receptors and external. After there are a cell for each receptors and externals, followed by the subcells. Each subcell is characterized by cell identifier (position 5), initial delay and period (positions 6 and 7), receptors (positions 8 and 9) and externals (position 10). The elements at positions 4, 8 and 9 are pointers: they indicates the location in which the values is contained. For example, the value 10 in the position 4 indicates that the external value of the pow2 cell at position 10. The external value of the main cell and the receptor of the subcells are always pointers.

The cellular space is in charge of managing the cell lifecycle: creation, updating and destroying. In the cellular space, in any moment of its life a cell is characterized by its *life-state*. As shown in Figure 5.12, there are 5 life-states for a cell:

- Created: a just created cell. When the cell will be added to the schedule, its state become Waiting or Frozen;
- Updating: in this state, the CPU is executing the update function of the cell. Only one cell can be in this state at any point in time, while all the other states can be adopted simultaneously by several cells,
- Waiting: all prerequisites for a transition into the running state exist, and the cell only waits for allocation of the processor;

- Frozen: in this state the cell is passive and can be activated or destroyed only by the other cells;
- Destroyed: a destroyed cell, waiting to be deleted from the memory.

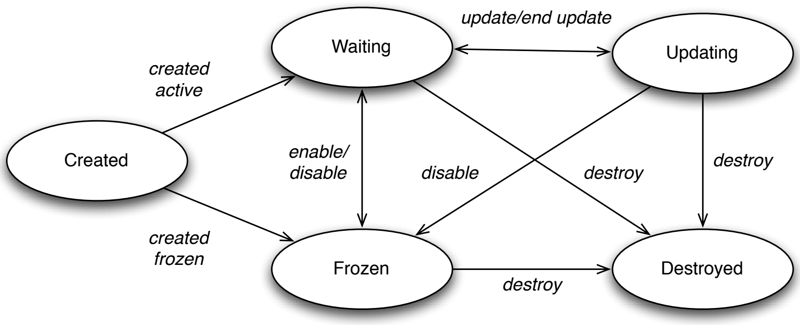


Figure 5.12: *The life-states of a cell.*

The Cellular Space is also the link between the cells and the external environment. In fact, through the cellular space, the cells interact with the device drivers and the dynamic behavior of the automata is influenced by the external environment and influences the external environment.

For the cells, the device drivers are "special" basic cells:

- *sensor cell* is a cell that can be forced from the external to change its state;
- *actuator cell* influences the external environment according to its external state;
- *open cell* is both a sensor cell and an actuator cell;

The MDCAee cellular space supports all the MDCA basic cell. Moreover it is possible to extend the basic cell defining (through the C language) new basic cells specific for the hardware and application.

There's a legal term for a problem in public space: something that might draw people to an area–say, across train tracks–where they might be caused harm. It's called a "public nuisance". I wouldnt mind being called that for my life's work.

Vito Acconci

6 The Indianapolis Project

In this chapter we describe an Adaptive Lighting system aimed at improving the everyday experience of pedestrians and people passing through the related environment.

A specific scenario related to the definition and development of an adaptive illumination facility is introduced, and a automata based model supporting the specified behavior for the illumination facility is defined. The presented system represents a first tentative step toward a cellular automata-based model for Adaptive Lighting.

A prototype of an environment supporting designers in the definition of the relevant parameters for this model and for the overall illumination facility is also introduced.

6.1 The Scenario

The Acconci Studio was founded in 1988 to help realize public-space projects through experimental architecture and public art efforts. The method of Acconci Studio is on the one hand to make a new space by turning an old one inside-out and upside-down; and on the other hand to insert within a site a capsule that grows out of itself and spreads into a landscape. They treat architecture as an occasion for activity; they make spaces fluid, changeable, portable. They have recently worked on a person-made island in Graz, a plaza in Memphis, a gallery in NY, a clothing store in Tokyo, a building façade in Milan, a park on a street median in Vienna, and a skate park in San Juan, Puerto Rico.

The Studio has been involved in a project for the renovation of a tunnel in the Virginia Avenue Garage in Indianapolis. The tunnel is currently mostly devoted to cars, with relatively limited space on the sidewalks and its illumination is strictly functional. Several photos of the Virginia Avenue Garage are shown in Figure 6.1.

The planned renovation for the tunnel comprises a set of interventions along the direction defined by the following narrative description of the project:

> The building bursts with color. Well no, not really: it might be more realistic to say that color spills out of the building, color oozes out of the

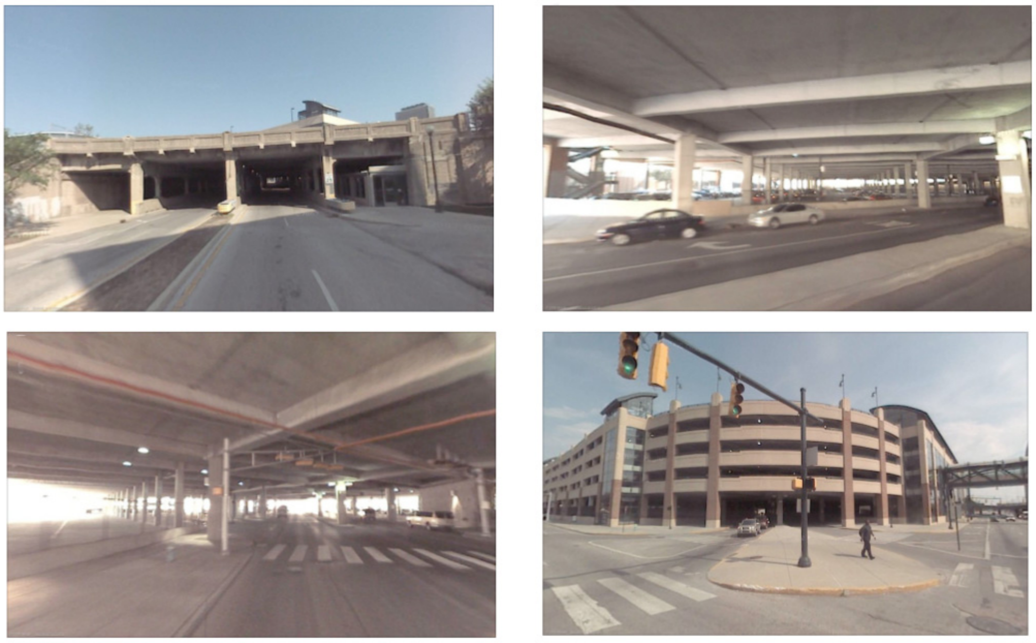


Figure 6.1: *Photos of the Virginia Avenue Garage, Indianapolis.*

building, color leaks out. The Studio has a disagreement here; Nate, who's spent some time in the garage, says that, during the day, so much sunlight comes in that any color would pale, white out. I don't want to believe that yet; I want to feel sure, at least for the time being, that – even if sunlight blots out the color at each end – there's enough of a middle that the passage through the building can hold its color, be its color.

The passage through the building should be a volume of color, a solid of color. It's a world of its own, a world in itself, separate from the streets outside at either end. Walking, cycling, through the building should be like walking through a solid, it should be like being fixed in color.

The color might change during the day, according to the time of day: pink in the morning, for example, becomes purple at noon becomes blue, or blue-green, at night. This world-in-itself keeps its own time, shows its own time in its own way.

The color is there to make a heaviness, a thickness, only so that the thickness can be broken. The thickness is pierced through with something, there's a sparkle, it's you that sparkles, walking or cycling though the passage, this tunnel of color. Well no, not really, it's not you: but it's you that sets off the sparkle – a sparkle here, sparkle there, then another sparkle in-between – one sparkle affects the other, pulls the other, like a magnet – a point of sparkle is stretched out into a line of sparkles is stretched out into a network of sparkles.

These sparkles are above you, below you, they spread out in front of you, they light your way through the tunnel. The sparkles multiply: it's you who sets them off, only you, but – when another person comes toward you in the opposite direction, when another person passes you, when a car passes by – some of these sparkles, some of these fire-flies, have found a new attractor, they go off in a different direction.

But these aren't fireflies. They're LED lights in a network, a mesh, a structure; the lights aren't moving with you, it's just that different lights are flashing in sequence, they go on-and-off linearly, radially, maybe they spiral as they go on-and-off... But the lights themselves aren't moving: it's as if the fireflies are pinned to their mesh of structure. Yes, you cause the lights; the lights aren't really swarming around you, following you, moving ahead of you and lighting your way – it's just that you're turning lights on-and-off as you move.

So we'd better be worried here; or at least we'd better be concerned. We know that you'll see the structure: the lights will light up the structure – the structure might be almost as visible as the lights. It's all a trick, and everybody can see through it. So does that mean we shouldn't be doing this, we shouldn't be trying to do what we know we can't do, we can't make the lights themselves move, like a swarm of insects, like a flock of baby birds...

Instead of doing what we can't do, let's do all we can. Instead of trying to hide the structure, let's admit the structure, and revel in the structure...

Picture a nighttime sky of branches: there's a jungle of branches, a bramble, a tangle of branches, above you and beside you. As you walk, as you cycle, something stirs in the branches: a swarm of fireflies is moving, lighting up, through the branches. But maybe you're not even thinking of branches anymore, don't think of fireflies: this night is blue-green, it's a purple night, a pink night – everything is more abstracted here and now. You're in a tangle, a ravel, a knot, the complex is swarming with particles of light, the particles of light are swarming around you...

The above narrative description of the desired adaptive environment comprises two main effects of illumination:

- An overall effect of uniformly coloring the environment through a background, ambient light that can change through time, but slowly with respect to the movements and immediate perceptions of people passing in the tunnel.

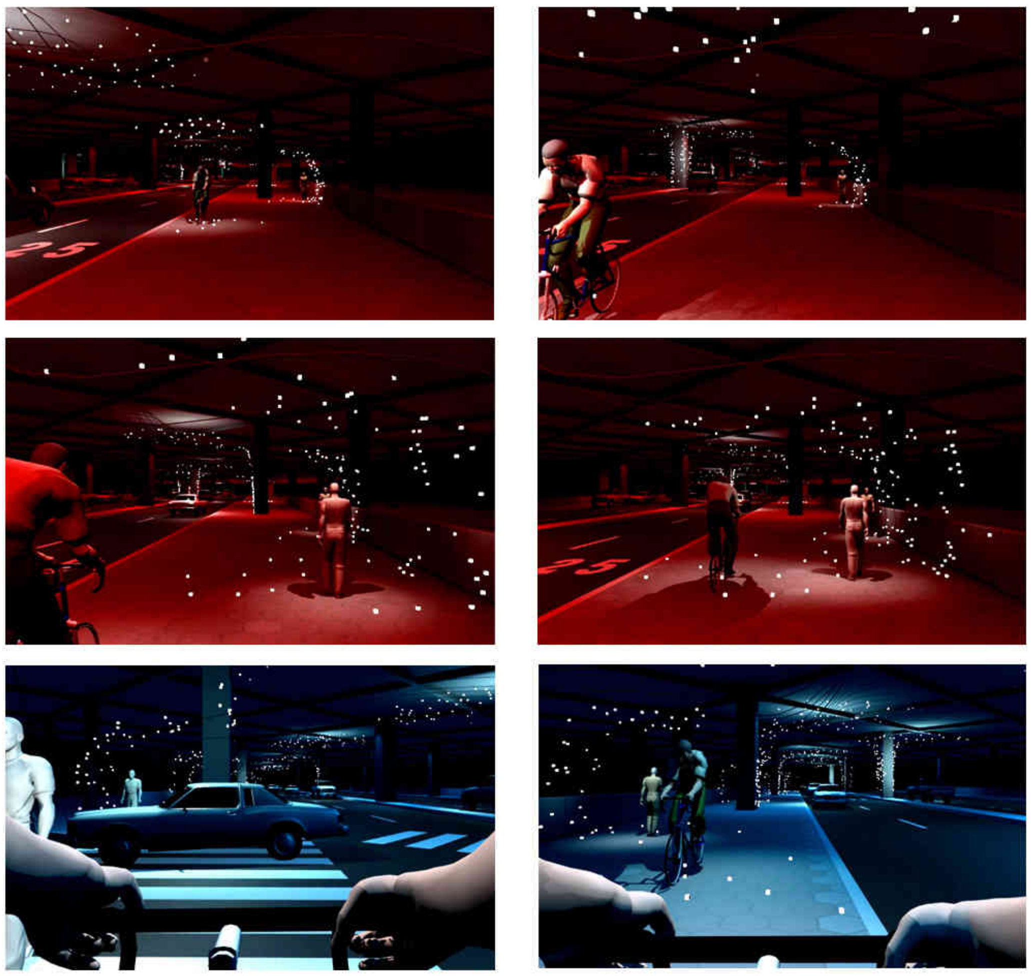


Figure 6.2: *Screenshots of a graphical animation showing the desired visual effects.*

- A local effect of illumination reacting to the presence of pedestrians, bicycles, cars and other physical entities, also depicted in a graphical elaboration of the desired visual effect shown in Figure 6.2.

The first type of effect can be achieved in a relatively simple and centralized way, requiring in fact a uniform type of illumination that has a slow dynamic. The second point requires instead a different view on the illumination facility. In particular, it must be able to perceive the presence of pedestrians and other physical entities passing in it, in other words it must be endowed with sensors. Moreover, it must be able to exhibit local changes as a reaction to the outputs of the sensors, providing thus for a non uniform component to the overall illumination. The overall environment must be thus split into parts, proper subsystems.

However, these subsystems cannot operate in isolation, since one of the require-

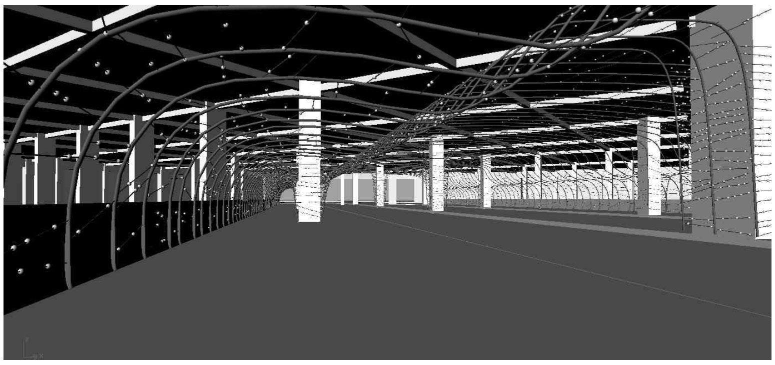


Figure 6.3: *Armature with lights viewed at eye level.*

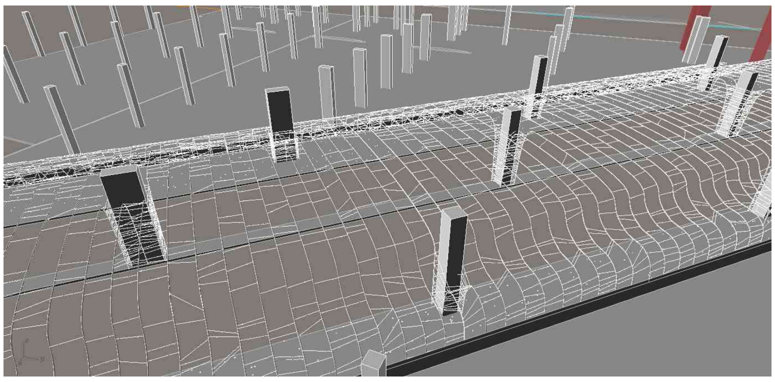


Figure 6.4: *Armature with lights viewed from above.*

ments is to achieve patterns of illumination that are local and small, when compared to the size of the tunnel, but that can have a larger extent than the space occupied by a single physical entity ("sparkles are above you, below you, they spread out in front of you, they light your way through the tunnel"). The subsystems must thus be able to interact, to influence one another to achieve more complex illumination effects than just providing a spotlight on the occupied positions.

6.2 A Network of Sensors and Actuators

The installation is composed of an armature that will hold up the swarm lighting. The armature, shown in Figures 6.3 and 6.4 is made of ribs of bent steel pipe, approximately 5 cm in diameter spaced at approximately 1.2 meters apart. The upper portion of these ribs is suspended from the concrete structure of the building.

Between these ribs, 1.2 cm diameter steel conduit serves a dual function of holding the LEDs and protecting the wiring. The wiring for the in air swarm lighting is channeled through the conduit and then though the ribs to a compartment that houses the

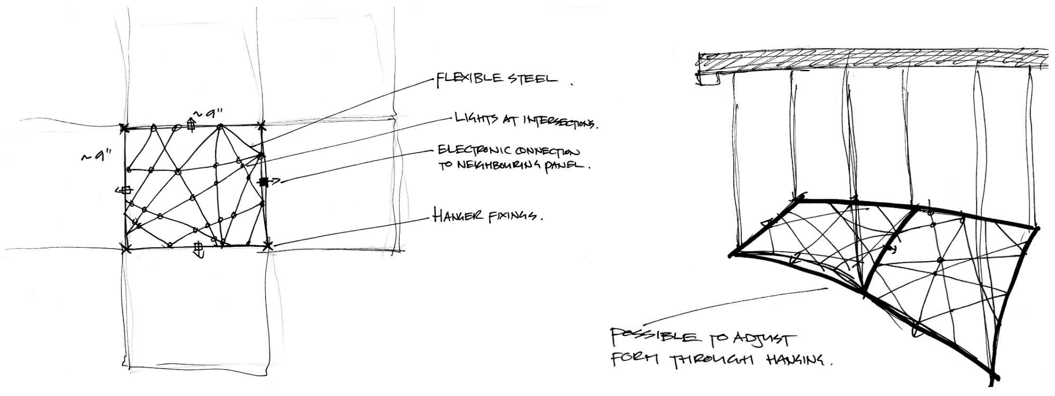

~9"
~9"
FLEXIBLE STEEL.
LIGHTS AT INTERSECTIONS.
ELECTRONIC CONNECTION TO NEIGHBOURING PANEL.
HANGER FIXINGS.
POSSIBLE TO ADJUST FORM THROUGH HANGING.

Figure 6.5: *A representation of the physical structure of a cell and the cells connections.*

microcontrollers and motion sensors.

The lighting system is created using networked cells, shown in Figure 6.5. To reduce installation time and price, these grids will be pre-fabricated off site and assembled on site later. These pre-fabricated cells will each house a microcontroller, a sensor, and a set of LED lights. Each cell has four communication line on the edges.

The designed system is an homogeneous peer system. As shown in Figure 6.6, every controller has the responsibility of managing the sensors and actuators belonging to a fixed area of the space. Controllers are homogeneous in terms of hardware and software capabilities. Every controller is connected to a motion sensor, which roughly covers the controlled area, some LEDs (about 40 LED lights) and neighboring controllers.

6.3 The Proposed Approach

The control of presented application can be achieved with a centralized control system (i.e. a central processor unit coordinates all the lights according to the state of the sensors) or a distributed control system. For this kind of that installation, a distributed control system is suitable and more natural, especially because the installation is composed of several components, with some degree of local control and global interactions.

Such control system can be schematized as in Figure 6.7. Each node of the system is composed of a control, a programmable device with an internal state (a persistence or volatile memory). A node can be connected to several sensors and actuators. Generally, in distribute control systems, the node are connected in a networks. Through the networks, the nodes exchange messages, in order to coordinate their activities.

The proposed approach adopts a Cellular Automata model to realize the distributed control system able to face the challenges of the previously presented scenario.

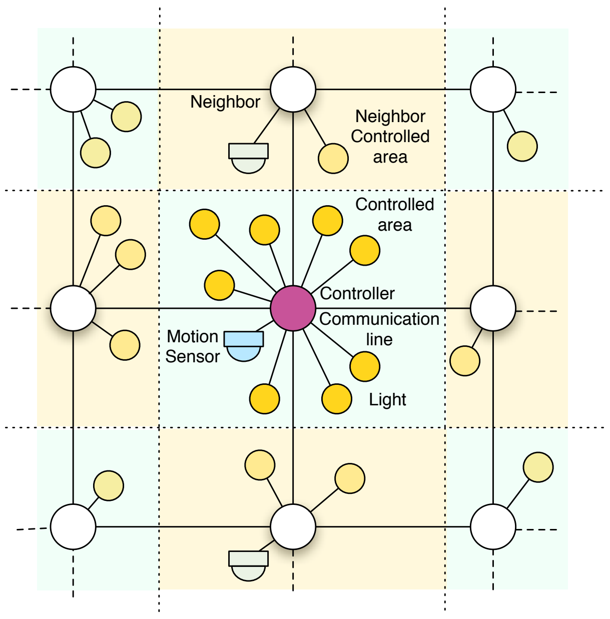


Figure 6.6: *A schematization of the architecture of the cells.*

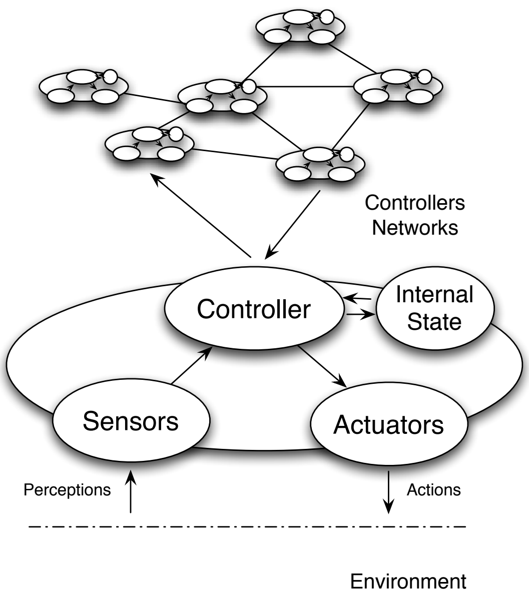


Figure 6.7: *Schematization of a distributed control system constituted of homogeneous nodes.*

In the proposed architecture, every node is a cell of an automata that can communicate only with its neighbors, processes signals from sensors and it controls a predefined set of lights associated to it. The approach is totally distributed: there is no centralized control and no hierarchical structuring of the controllers, nor from a logical point of view neither from a physical one.

In the following sections, each of the components of the proposed approach will be described in details.

6.3.1 System Architecture

The proposed architecture is composed of four layers:

- inter-controller communication layer;
- intra-controller layer;
- sensors layer;
- actuators layer.

As shown in Figure 6.8, the external layer (level 2) is the communication layer between the controllers of the system. Each controller is an automata network of two nodes. One node is a sensor communication layer and represents a space in which each sensor connected to the microcontroller has a correspondent cell. The other node represents the actuators layer in which the cells pilot the actuators (lights, in our case). Since the external layer is a physical one and every cell is an independent microcontroller, it cannot be assumed that the entrire network is synchronized. In same cases, a synchronous network can be constructed (for example, a single clock devices can be connected to each microcontrollers or the microcontrollers can be synchronized by a process without a master node), but the most general case is an asynchronous network.

6.3.2 Sensors Layer

The Sensor Layer is a Level 0 Dissipative Automata. As previously introduced, it is composed of a single cell, since only one sensor is connected to each microcontroller. It is a Dissipative Automata because the internal state of the cell is influenced by the external environment. The state of the cell is represented by a single numerical value $v_s \in \mathbb{N}_{8bit}$, where

$$\mathbb{N}_{8bit} \subset \mathbb{N}_0, \forall x : x \in \mathbb{N}_{8bit} \Rightarrow x < 2^8 \tag{6.1}$$

The limit value was chosen for performance reasons because 8-bit microcontrollers are widely diffused and they can be sufficiently powerful to manage this kind of situation. The value of v_s is computed as

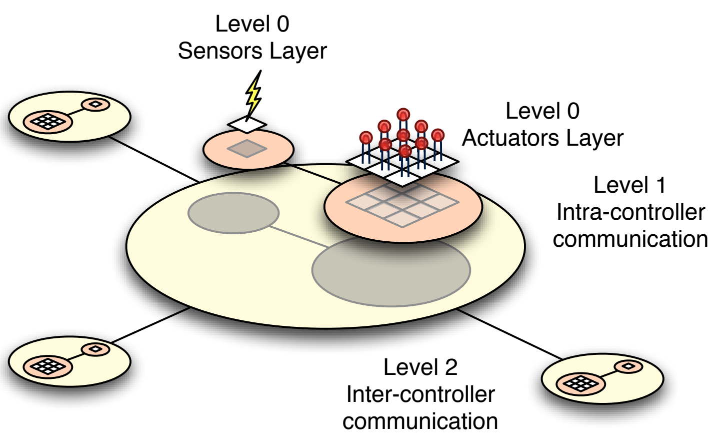


Figure 6.8: *The proposed automata network.*

$$v_s(t+1) = v_s(t) \cdot m + s(t+1) \cdot (1-m) \tag{6.2}$$

where $m \in \mathbb{R}, 0 \leq m \leq 1$ is the *memory coefficient* that indicates the degree of correlation between the previous value of v_s and the new value, $s(t) \in N_{8bit}$ is the reading of the sensor at the time *s(t)*. If the sensor is capable of distance measuring, *s(t)* is inverse proportional to the measured distance (so, if the distance is 0, the value is 255, if the distance is ∞ the value is 0). If the sensor is a motion detector sensor (it able to signal 1 if an object is present or 0 otherwise) *s(t)*, s(t) is equal to 0 if there is not detected motion, *c* in case of motion, where $c \in \mathbb{N}_{8bit}$ is a constant (in our tests, 128 and 192 are good values for *c*).

6.3.3 Diffusion Rule

In this section we describe the diffusion rule, that is used to propagate the sensors signals through the system. At a given time, every level 2 cell is characterized by an intensity of the signal, $v \in \mathbb{N}_{8bit}$. Informally, the value of v at time $t+1$ depends of the value of v at time t and on the value of $v_s(t+1)$, to capture both the aspects of interaction with neighboring cells and the memory of the previous external stimulus caused by the presence of a physical entity in the area associated to the cell.

The intensity of the signal decreases over time, in a process we call evaporation. In particular, let us define $\epsilon_{evp}(v)$ as the function that computes the quantity of signal to decrement from the signal and is defined as

$$\epsilon_{evp}(v) = v \cdot e_1 + e_0 \tag{6.3}$$

where $e_0 \in \mathbb{R}^+$ is a constant evaporation quantity and $e_1 \in \mathbb{R}, 0 \leq e_1 \leq 1$ is the evaporation rate (e.g. a value of 0.1 means a 10% evaporation rate).

The evaporation function $evp(v)$, computing the intensity of signal v from time t to $t+1$, is thus defined as

$$evp(v) = \begin{cases} 0 & \text{if } \epsilon_{evp}(v) > v \\ v - \epsilon_{evp}(v) & \text{otherwise} \end{cases} \tag{6.4}$$

The evaporation function is used in combination with the neighbors' signal intensities to compute the new intensity of a given cell. We first present the formula for a regular neighborhood and than we generalize to the irregular structure.

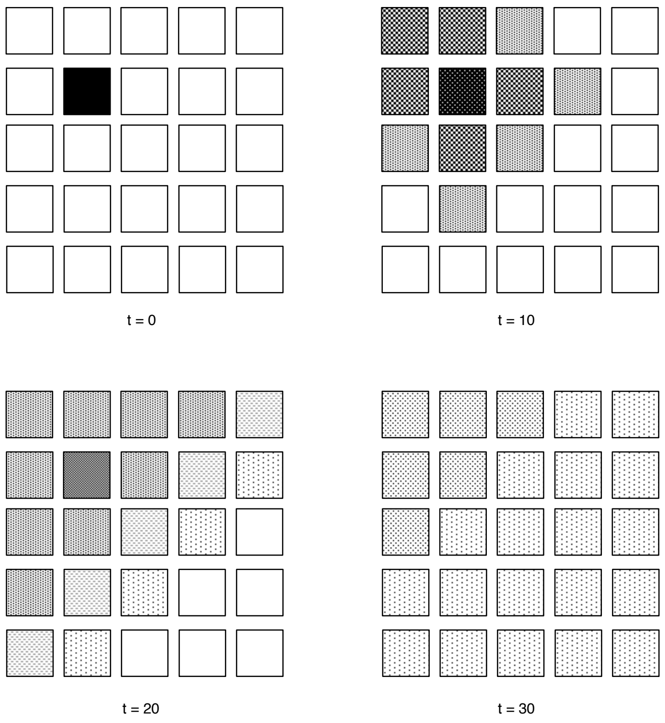


Figure 6.9: *An example of the dynamic behavior of a diffusion operation. The signal intensity is spread throughout the lattice, leading to a uniform value; the total signal intensity remains stable through time, since evaporation was not considered.*

6.3.3.1 Regular neighborhood

The automaton is contained in the finite two-dimensional square grid $\mathbb{N}^2$. We suppose that the cell $C_{i,j}$ is located on the grid at the position i, j, where $i \in \mathbb{N}$ and $j \in \mathbb{N}$. According to the von Neumann neighborhood (Gutowitz, 1991), a cell $C_{i,j}$ (unless it is placed on the border of the lattice) has 4 neighbors (as shown in figure 6.10), denoted by $C_{i-1,j}$, $C_{i,j+1}$, $C_{i+1,j}$, $C_{i,j-1}$. For simplicity, we numbered the neighbors of a cell from 1 to 4, so for the cell $C_{i,j}$, N_1 is $C_{i-1,j}$, N_2 is $C_{i,j+1}$, N_3 is $C_{i+1,j}$, and N_4 is $C_{i,j-1}$

At a given time, every cell is characterize by an intensity of the sensor signal. Each cell is divided into four parts (as shown in Figure 6.10), each part can have a different signal intensity, and the overall intensity of the signal of the cell is the sum of the parts intensity values. The state of each cell $C_{i,j}$ of the automaton is defined by $C_{i,j} = \langle v_1, v_2, v_3, v_4 \rangle$ where $v_1, v_2, v_3, v_4 \in \mathbb{N}_{8bit}$ represent the intensity of the signal of the 4 subparts. $V_{i,j}(t)$ represents the total intensity of the signals (i.e. the sum of the subparts' signal intensity) of the cell i, j at time t. The total intensity of the neighbors are denoted by V_{N1}, V_{N2}, V_{N3}, and V_{N4}. The signal intensity of the subparts and the total intensity are computed with the following formulas:

$$v_j(t+1) = \begin{cases} \frac{evp(V(t)) \cdot q + evp(V_{Nj}(t)) \cdot (1-q)}{4} & \text{if } \exists N_j \\ \frac{evp(V(t))}{4} & \text{otherwise} \end{cases} \tag{6.5a}$$

$$V(t+1) = \sum_{i=1}^{4} v_i(t+1) \tag{6.5b}$$

where $q \in \mathbb{R}, 0 \leq q \leq 1$ is the conservation coefficient (i.e. if q is equals to 0, the new state of a cell is not influenced by the neighbors' values, if it is equals to 0.5 the new values is a mean among the previous value of the cell and the neighbors' values, if it is equals to 1, the new value does not depend on the previous value of the cell but only from the neighbors). The effect of this modeling choice is that the parts of cells along the border of the lattice are only influenced through time by the contributions of other parts (that are adjacent to inner cells of the lattice) to the cell's intensity.

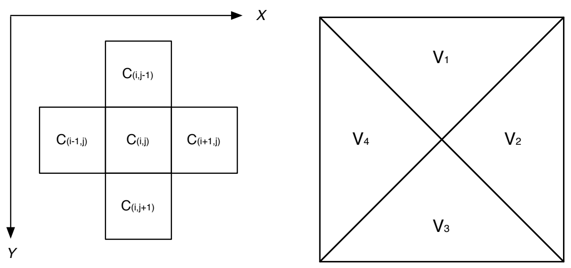


Figure 6.10: *On the left, the von Neumann neighborhood of the cell $C_{i,j}$, on the right, the internal structure of a cell of the regular automaton.*

6.3.3.2 Irregular neighborhood

The irregular structure automata is a generalization of the regular one. The automaton is composed of cell numbered from 1 to N, so we use C_i for $0 \leq i \leq N$ to indicate the i-th cell. Every cell C_i can have an arbitrary number of neighbors $L_i, 0 \leq L_i \leq L \leq N - 1$ where L_i is the numbers of neighbors of the cell C_i and $L = \max(L_i)$ is the maximum numbers of neighbors of every cell the system. Neighboring cells of cell i can be denoted as $N_{i,l}$.

As for the regular neighborhood case, each cell is divided into L parts, each part can have a different signal intensity, and the overall intensity of the cell's signal is the sum of the intensity values of the parts.

The state of each cell C_i of the automaton is defined as $V_i = \sum_{l=0}^{L_i} v_{i,l}$ where $v_{i,l} \in \mathbb{N}_{8bit}$ represent the intensity of the signal of the L subparts. Finally, the intensity of each neighboring cell of C_i is denoted by $V_{i,l}$.

The signal intensity of the subparts and the total intensity can thus be computed according to the following formulas:

$$v_{i,l}(t+1) = \begin{cases} \frac{evp(V_i(t)) \cdot q + evp(V_{i,l}(t)) \cdot (1-q)}{L} & \text{if } \exists N_{i,l} \\ \frac{evp(V_i(t))}{L_i} & \text{otherwise} \end{cases} \tag{6.6a}$$

$$V_i(t+1) = \sum_{l=1}^{L_i} v_{i,l}(t+1) \tag{6.6b}$$

In the real system, the maximum number of neighbors (L) is constrained by the number of available inputs on the microcontrollers.

6.3.4 Actuators Layer

The cells of the actuator layer determinate the actuators actions. In this project the actuators are LED lamps that are turned on and of according the the state of the cell. Instead of controlling a single LED from a cell, every cell is related to a group of LEDs disposed in the same (small) area.

In the case of regular neighborhood, each controlled area in divided into 9 sub-areas and each sub-area contains a group of LEDs controlled by the same actuators layer cell. The state of each cell is influenced only by the state of the signal intensity of the upper layer cell. The correlation between the upper layer cell subparts and the actuators layer cells is shown in Figure 6.11.

The state of the actuators cells $A_1..A_9, A_j \in N_{8bit}$ is computed with the following formula:

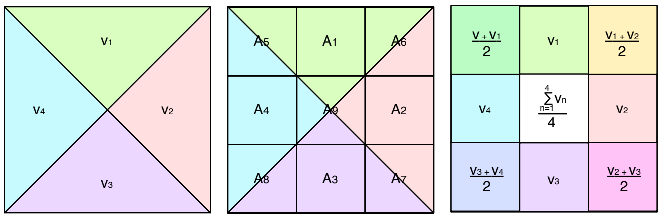


Figure 6.11: *Correlation between the upper layer cell subparts and the actuators layer cells.*

$$A_i(t+1) = \begin{cases} v_i(t+1) & 1 \le i \le 4 \\ \dfrac{v_4(t+1) + v_1(t+1)}{2} & i = 5 \\ \dfrac{v_1(t+1) + v_2(t+1)}{2} & i = 6 \\ \dfrac{v_2(t+1) + v_3(t+1)}{2} & i = 7 \\ \dfrac{v_3(t+1) + v_4(t+1)}{2} & i = 8 \\ \dfrac{1}{4}\displaystyle\sum_{j=1}^{4} v_j(t+1) & i = 9 \end{cases} \tag{6.7}$$

There are different approaches, called "coloring strategies", to associate LED activity (i.e. being on or off, with which intensity) to the state of the related actuator cell. An example of coloring strategy consists in directly connecting the lights' intensity to the signal level of the correspondent cell; more details on this will be given in the following Section.

6.4 The Design Environment

The design of a physical environment (e.g. building, store, square, road) is a composite activity, comprising several tasks that gradually define the initial idea into a detailed project, through the production of intermediate and increasingly detailed models. CAD softwares, and also 3D modeling applications are generally used to define the digital models for the project and to generate photo realistic renderings and animations. These applications are extremely useful to design a lights installation like the one related to this scenario, but mainly from the physical point of view.

In order to generate a dynamics in this kind of structure, to grant the lights the ability to change illumination intensity and possibly color, it is also possible to "script" these applications in order to characterize lights with a proper behavior. Such scripts, created as text files or with graphical logic editors, define the evolution of the overall system over time. These scripts are however heavily dependent on the adopted software and they are not suitable for controlling real installations, even though they can be used to achieve a graphical proof of concept. Another issue is that these tools are characterized by a “global” approach, whereas the system is actually made up of individual microcontroller program acting and interacting to achieve the global desired effect.

In this experience, our aim was to facilitate the user in designing the dynamic behavior of a lights installation by supporting the envisioning of the effects of a given configuration for the transition rule guiding lights; therefore we created an ad-hoc tool, also shown in Figure 6.13, comprising both a simulation environment and a graphical parameters configurator. This tool support the specification of the values for some of the parameters of the transition rule, affecting the global behavior of the overall system. The integrated simulation helps understanding how the changes of the single parameters influence the overall behavior of the illumination facility: every changed parameter is immediately used in the transition rule of every cell.

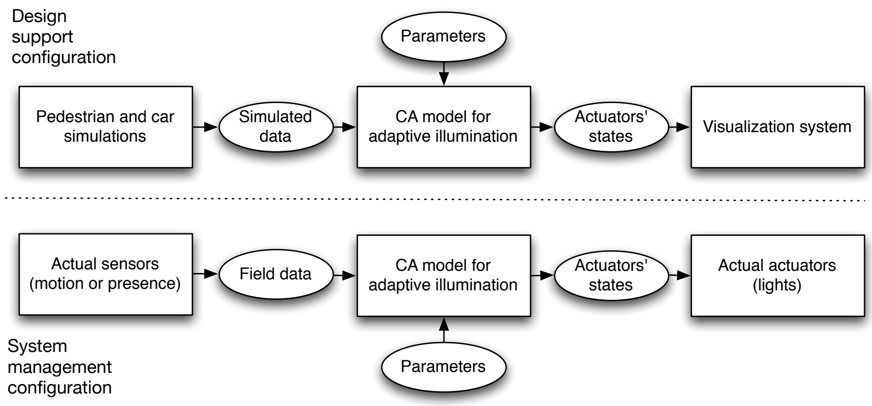

Design support configuration
Parameters
Pedestrian and car simulations
Simulated data
CA model for adaptive illumination
Actuators' states
Visualization system
Actual sensors (motion or presence)
Field data
CA model for adaptive illumination
Actuators' states
Actual actuators (lights)
System management configuration
Parameters

Figure 6.12: *A schema describing the modules of the design support system prototype.*

The design environment is composed of two main modules, also shown in Figure 6.12: a simulation environment (that is in turn decomposed into a pedestrian and cars simulation module and an adaptive lighting module) and a visualization facility. In the following paragraphs these modules will be described.

6.4.1 The Simulation Environment

The simulation environment actually comprises three main modules:

- lights simulator;
- pedestrians simulator;
- cars simulator.

The *light simulator* simulates the network of controllers with sensors and actuators. The *pedestrians and cars simulator* simulates the environment in which the adaptive

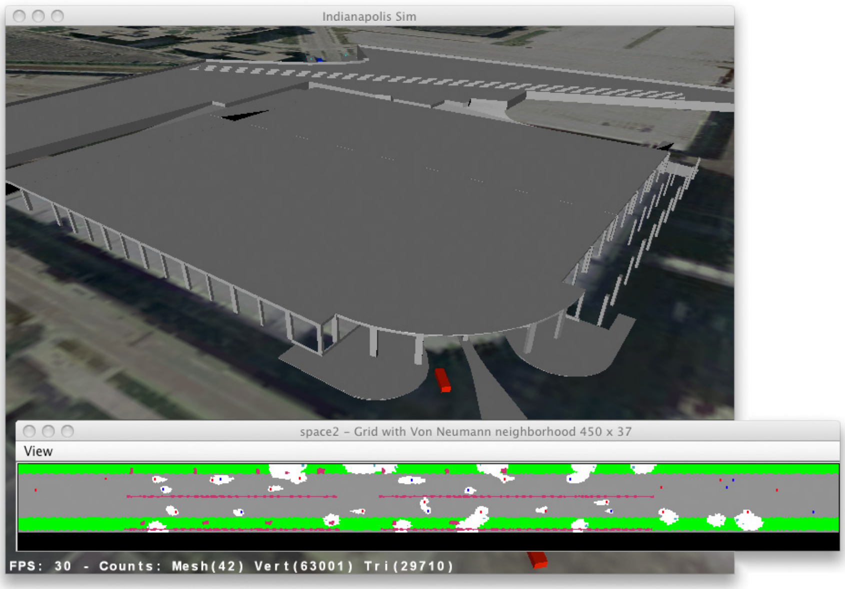

Indianapolis Sim
space2 - Grid with Von Neumann neighborhood 450 x 37
View
FPS: 30 - Counts: Mesh(42) Vert(63001) Tri(29710)

Figure 6.13: *Screenshot of the design environment. The front windows is the 2D visualization of the pedestrian, car an lights simulation. The background windows is the 3D view of the environment, including the architectural structures, lights, pedestrians the cars.*

lighting facility is situated and the pedestrians and cars situated in it. The two simulations are connected: in particular, the state of the sensors is influenced by the position of the simulated pedestrians and cars.

The *lights simulator* simulates the dynamic evolution of the cells over the time, according to the transition rule. In order to simulate an asynchronous system, an independent thread of control, that re-evaluates the internal state of the cell every 200 ms is associated to each cell. At the simulation startup, each thread starts after a small (< 1 s) random delay, in order to avoid a sequential activation of the threads, that is not realized in the real system. The operating system scheduler introduces additional random delays during both the activation and the execution cycle of the threads.

The pedestrian and car simulator is based on the MMASS (Bandini et al., 2006) model. This module actually feeds the self-organization model with simulated field data. The previously described model managing the self-organization of the illumination facility will react according to the current occupation of the space in the environment and according to its own parameters.

In this way, the designer can effectively envision the interaction between the people an the specified adaptive environment. The simulation environment allows the designer in configuring the network, defining the type, number, position of the sensors and actuators, and in specifying the behavior of the controllers, by means of defining the parameters of the model.

6.4.2 The Visualization Facility

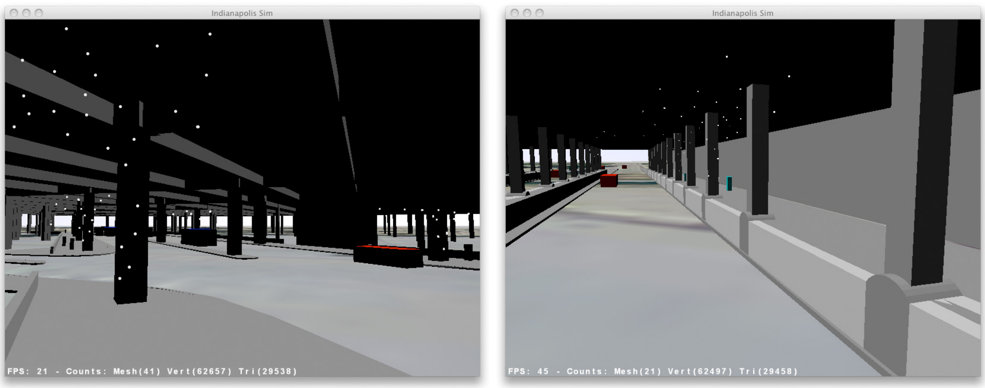
The design environment provides both 2D and 3D views. The 2D visualization is interactive: it is possible to define an action event to be fired when an object is clicked (e.g. simulate the perception of a pedestrian when the user clicks on a sensor). This is useful because allows the designer to test the system behavior before specifying in an extensive way a pedestrian simulation scenario.

The 3D visualization is useful not only to "understand" the global behavior of the system but also to visualize how the pedestrian and the car drivers perceive the lights system. The 3D visualization is based on the jMonkey engine[1],an open source 3D engine written in Java. The engine simplifies the development of 3D applications. It allows loading several 3D model formats and supports many high level effects (e.g. lens flare, particle systems).

In the 3D view, there are three different modes of exploration of the space:

- free look;
- pedestrian's viewpoint;
- car driver's viewpoint.

During the simulation, the user can switch between the modes with the keyboard. In the *free look* mode, the user can freely navigate the 3D space with the mouse, changing his/her point of view. He/she is not subjected to the gravity force, so he/she can move up/down and observer the simulation from an arbitrary point of view, as shown in Figure 6.13.



FPS: 21 - Counts: Mesh(41) Vert(62657) Tri(29538)

FPS: 45 - Counts: Mesh(21) Vert(62497) Tri(29458)

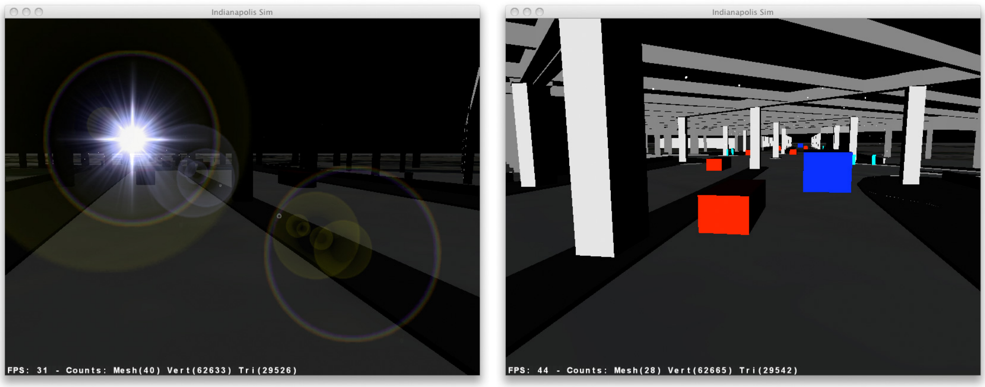
Figure 6.14: *Two screenshots of the 3D view: on the left, an example of pedestrian's viewpoint, on the right, car driver's viewpoint.*

In the *pedestrian's viewpoint* mode, the user takes the perspective of one of the pedestrians walking in the environment. The user cannot move freely because the

[1]http://www.jmonkeyengine.com/

position is given by the simulated pedestrian, but it can "rotate the head" with the mouse. The *car driver's viewpoint* is similar but the viewpoint of is centered on a car driver, as shown in Figure 6.14.

The 3D visualization system also simulates the position of the sun during the different hours of the day, as shown in Figure 6.15. It is important for the design of the lights system because the amount of environment light influence the person ability to see the lights effects. The daylight simulation is based on the model presented in Preetham et al. (1999).



```
FPS: 31 - Counts: Mesh(40) Vert(62633) Tri(29526)
FPS: 44 - Counts: Mesh(28) Vert(62665) Tri(29542)
```

Figure 6.15: *Two screenshots of the 3D view representing the environment in the late evening.*

An amazing invention - but who would ever want to use one?

made a call from Washington to Pennsylvania with Alexander Graham Bell's telephone, patented on 7 March 1876

7 From Theory to Product: Digital Footprints

7.1 Toward a Modular Adaptive Lighting System

IN this chapter we propose a Modular Adaptive Lighting System composed of several independent modules. Each module is an Adaptive Lighting system, equipped with proximity sensors, and RGB leds.

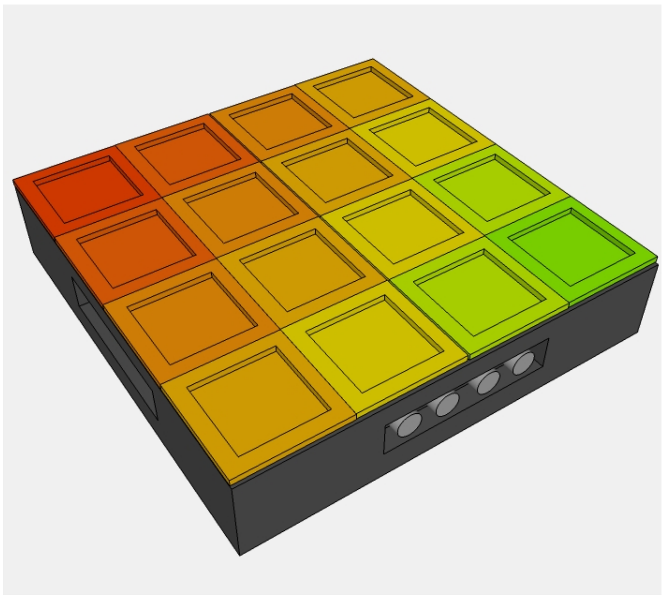


Figure 7.1: *A rendering of the proposed Adaptive Lighting module.*

A module can be used alone or together with other modules. The modules are square shaped and they can be combine like "bricks" in order to create a larger lighting system. The modules can be combined in several different ways, as shown in Figure 7.5. For example the modules can be arranged around a window, they can be used to create an aesthetic composition on a wall, they can be put along a passage or on the floor.

An existing product, similar to the proposed one, is Color Kinetics® iColor® Mod-

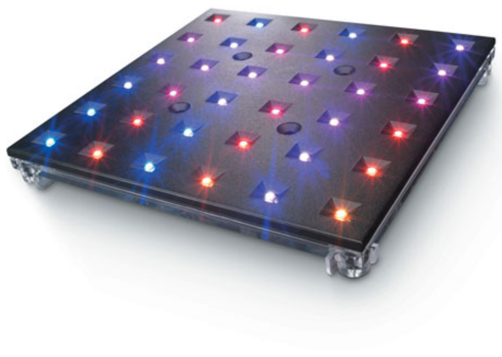


Figure 7.2: *Color Kinetics® iColor® Module FX with 36 leds (15.2 cm x 15.2 cm).*

ule FX[1]. The module, shown in Figure 7.2, is a 15.2 cm square modular unit intended for versatile designs that call for individually controllable points of light, including intricate patterns, images, animation and video. Each panel incorporates 9 or 36 individually addressable RGB leds on a circuit board employing a microchip that integrates power, communication, and control to individual nodes, across one unit or a multi-unit installation. The modules are compatible with most Color Kinetics controllers and third-party DMX controllers. This system is similar to the proposed one, however there are a deep difference: these panels do not include any sensors and are intended to be controlled by as a display device by and external centralized controller.

Helen Evans and Heiko Hansen made a modular light system for architecture called Light Brix, shown in Figure 7.3. It is a modular light system that responds to touch: through the electromagnetic fields of the human body. The hexagonal units can be assembled in any shape and modulated to compose multiple lighting situations.

The MIT Media Lab developed a set of *smart blocks*, shown in Figure 7.4, called "siftables" (Merrill et al., 2007) (Ullmer and Schmidt, 2007). The blocks are small, self-contained input and display devices wirelessly link together to form an independent network. Users feed information into the system simply by shuffling the siftables around, while the siftables themselves relay information back to the user. Each siftable measures about 5-centimetres square and is fitted with an LCD screen, battery, memory, an accelerometer to detect motion, a Bluetooth radio to communicate with other computers, and an infrared link to detect the presence and orientation of neighboring siftables.

Siftables are wireless battery power devices that can be easily recombined, the proposed modular Adaptive Lighting device can be assembled in the desired shaped during the installation (or the reconfiguration) of the system. Depending on the type of

[1]Color Kinetics and iColor are registered trademarks of Color Kinetics Incorporated.

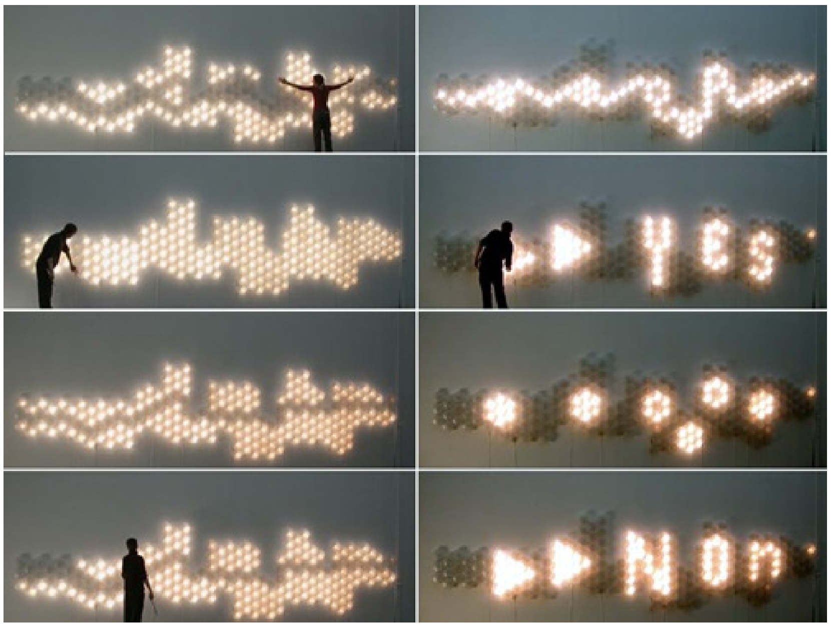


Figure 7.3: *Light Brix by Helen Evans and Heiko Hansen.*

installation, the modules can be connected each others only by magnetic force or they can be installed on a frame. As shown in Figure 7.1, each module has power and communication lines on its edge: when two modules are side by side, they communicate together.

7.2 Hardware Prototype

In 2009 CSAI (Complex System and Artificial Intelligence Research Centre) started a collaboration with Egicon (stratEGIC innovatiON)[1], an italian electronic engineering company that has been founded by people with many years of experience in electronic business. Egicon mission is to support the customer with innovative and effective solutions from the design to the production. Egicon wants to be the strategic partner for innovation.

The aim of the collaboration between CSAI an Egicon is the development of a prototype of a hardware platform suitable for a modular Adaptive Lighting. The prototypal board developed by the Egicon engineers has the following components:

- 16 RGB leds
- 16 proximity sensors
- a speaker

[1]http://www.egicon.com

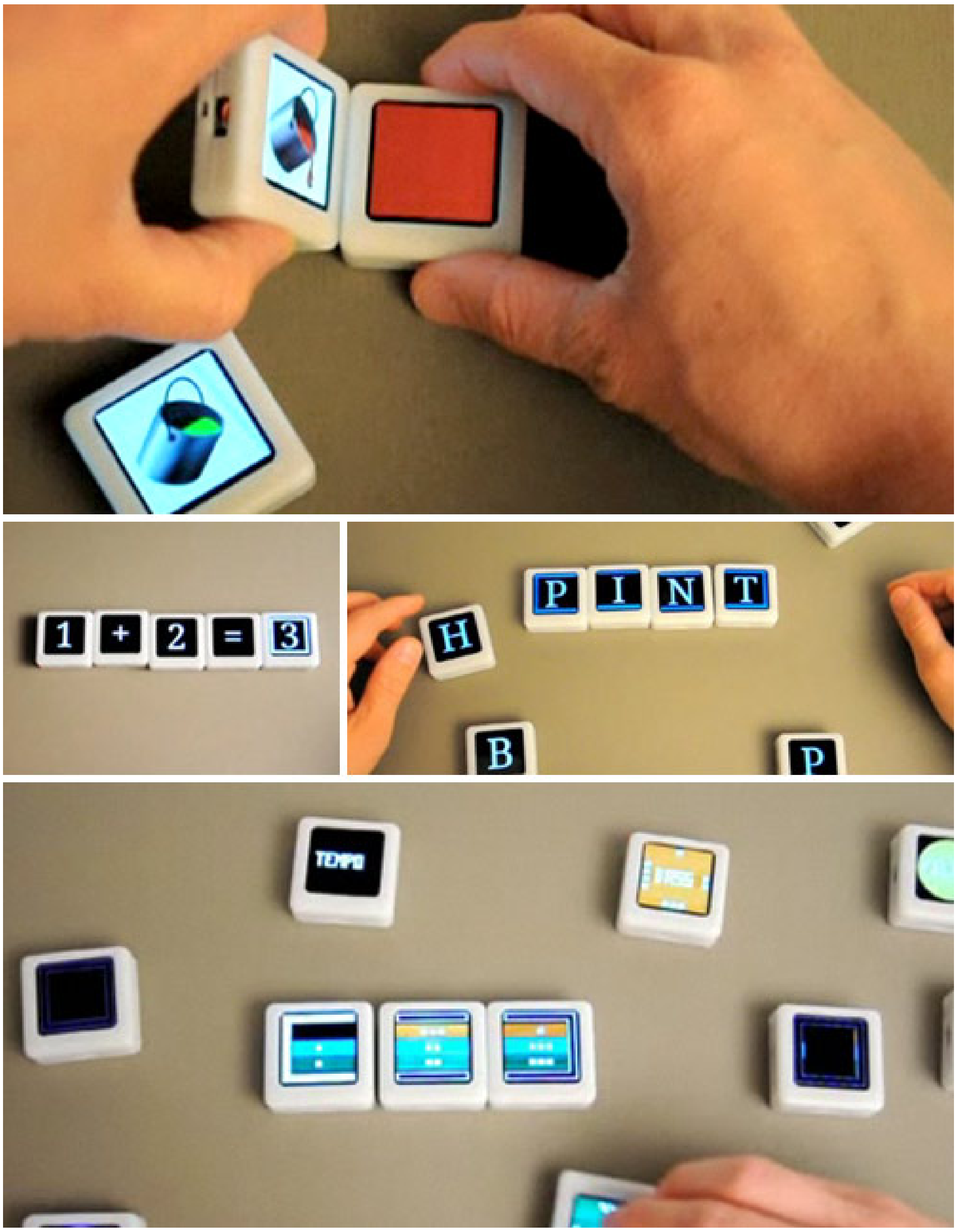


Figure 7.4: *MIT Media Lab Siftables (5 cm x 5 cm).*

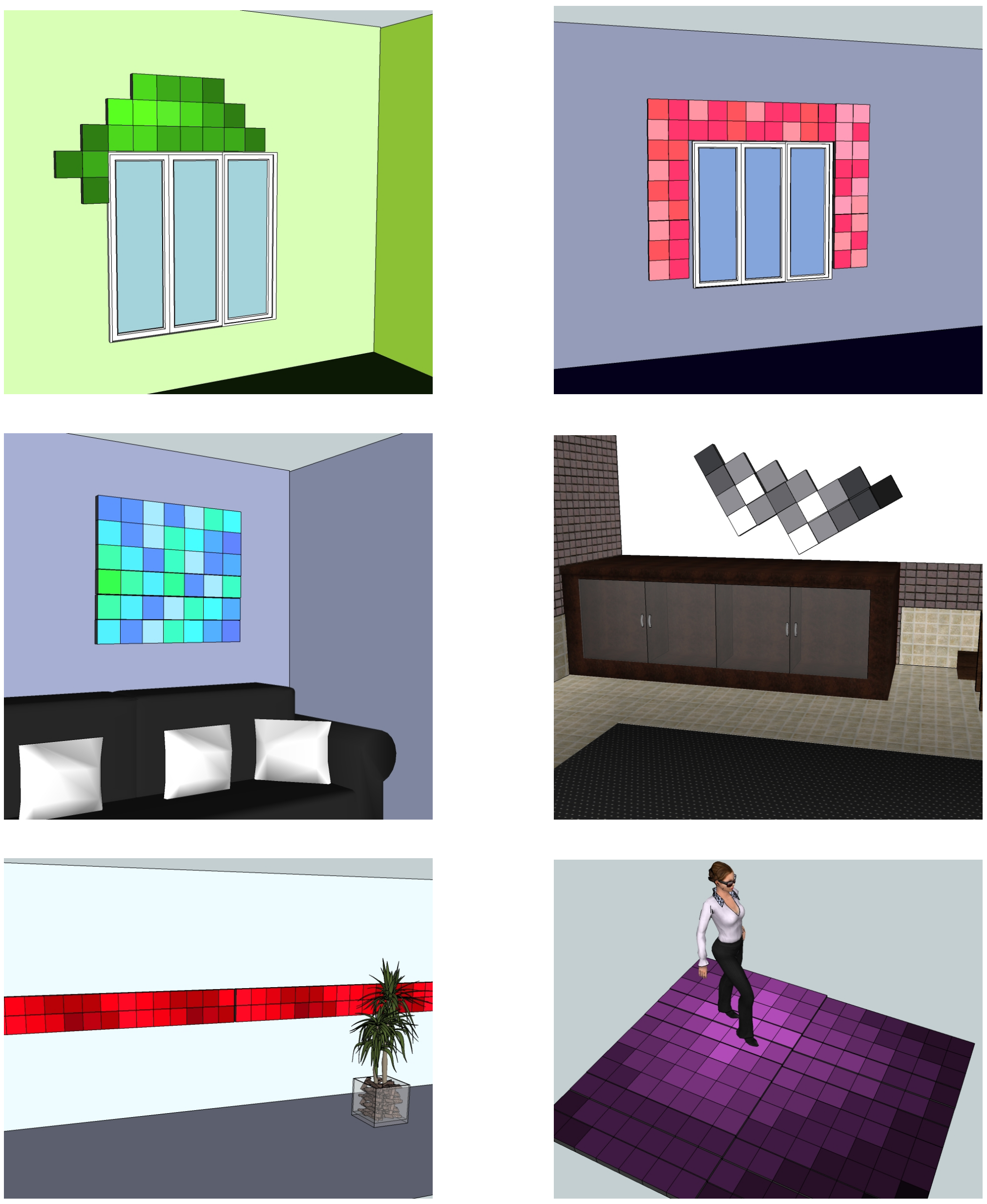


Figure 7.5: *Examples of different compositions of the modular Adaptive Lighting system.*

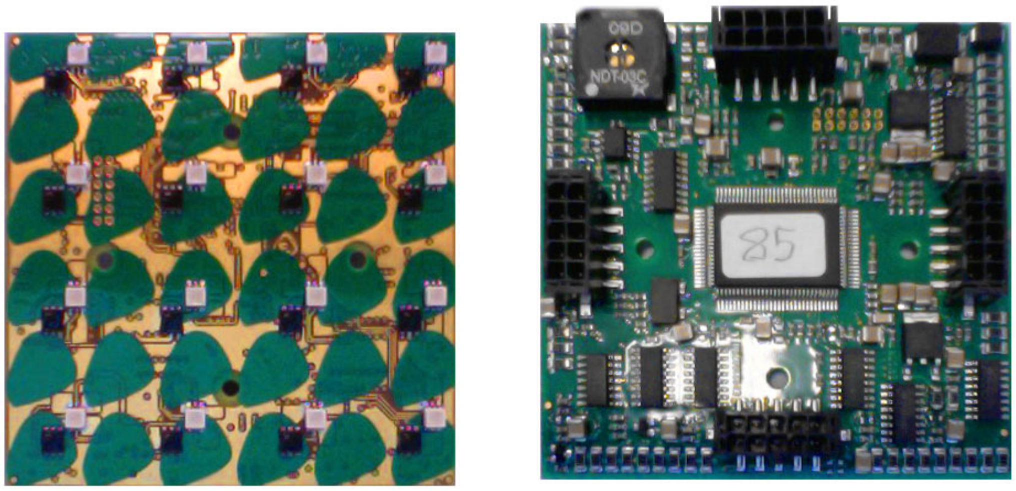


Figure 7.6: *The front and back sides of the module prototype. In the front picture, the white elements are the RGB leds, the black elements are the proximity sensors. In the back picture, the central element is the Fujitsu F^{2}MC-16LX microcontroller, on the edges there are the connector, on top left the speaker.*

- 4 communication and power connector
- a Fujitsu F^{2}MC-16 microcontroller

The RGB led contains three leds (red, green and blue) encased in one shell. It looks like a single white led except that it has four leads - one for the common ground connection and one for each led. The current through each of the leds determines its light output (i.e. its contribution to the total output color). By controlling the current through each led it is possible to obtain different light colors.

The proximity sensors are optical proximity switch that reacts at a typical working distance of 20 mm. The sensors allow the users to interact with the Adaptive Lighting system simply touching the glass covering the lights, without pushing any buttons.

As depicted in Figure 7.7, the board has 4 connectors on the edges. Each connector carries both a bi-directional serial communication line and a power line, so only one of the module of an Adaptive Lighting system need to be directly connected to the power supply. Moreover each connectors provide a shared one-wire serial line for communication with an external system (e.g. a pc). This communication line is used to send command from an external controller to all the boards (e.g. for reprogramming the boards).

The board is driven by a Fujitsu F^{2}MC-16LX microcontroller. The family of F^{2}MC-16LX series microcontrollers serve for consumer (e.g. digital cameras, handheld electronic product), white goods (e.g. washing machines, refrigerators), industrial (e.g. utility meters, air-conditioning systems), and automotive (e.g. body control networks,

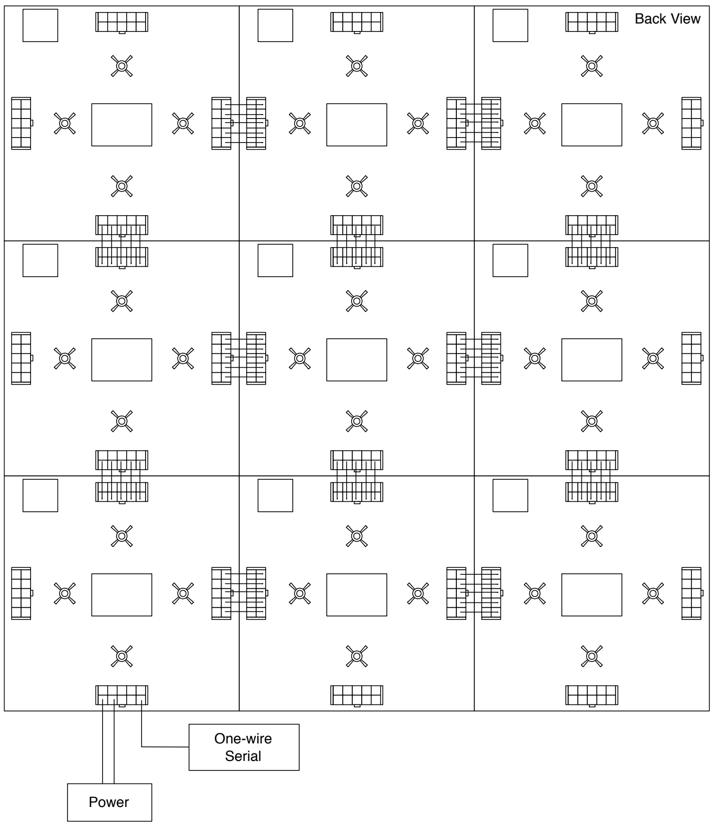


Figure 7.7: *Schematization of the connections between 9 modules.*

dashboards, chassis networks) applications. The MB90347 microcontroller is a 16 bit CISC running at 24MHz and has 128Kb Flash and 6Kb RAM.

7.3 Software Implementation

The prototype is a MDCA Embedded Environment (MDCAee) comprising cells of different types, as depicted in Figure 7.8. The main cells types are:

- Body
- Actuator
- Edge
- Command Shell

In the rest of this section we describe we describe the default behavior of such cells. Moreover is possible to reprogram such cells in order to obtain different behaviors.

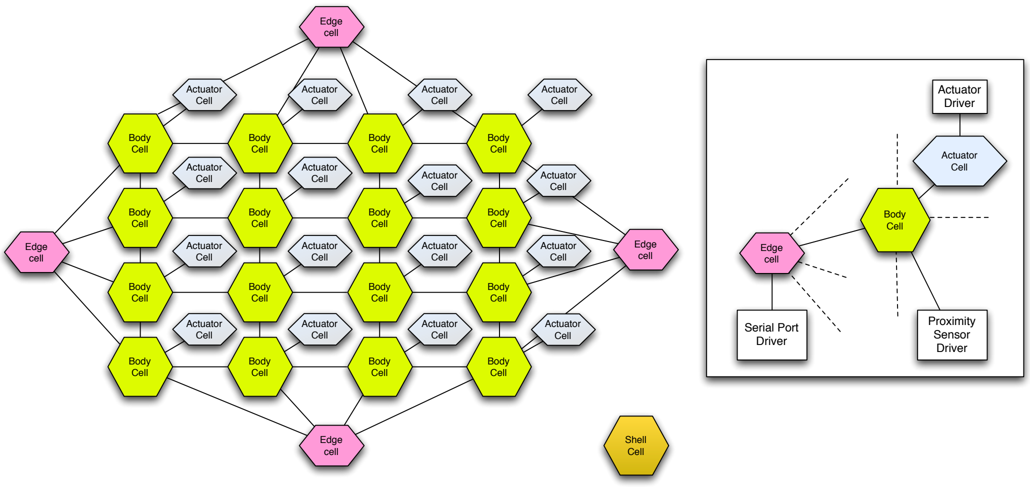


Figure 7.8: *On the left, the schematization of the module cells. The connections between the* Shell Cell *and the other cells are not shown for simplicity. On the right, a detail about the connection between the cells and the drivers.*

7.3.1 Body Cell

Each body cells is characterized by three "substances levels". To introduce a similarity with the biological cell, each substance level represent the amount of a specific chemical substance inside the cell (e.g. CA^{++}). The three "virtual substances" are named Q_1, Q_2 and Q_3. Internally, the levels of the three substance (respectively s_1, s_2 and s_3) are represented by an integer number between 0 and $2^{16}-1$.

The amount of each substance is influenced by three processes:

- Stimulus Response
- Evaporation
- Diffusion

The *Stimulus Response* is the reaction of the cell to an external stimulus. The body cells are able to react to external stimuli because they are connected to the proximity sensors. Each body cell is connected to a different sensor. When a sensor is stimulated, the levels of the virtual substance of the related cell are increased in response to the stimolus. Each substance level $s_i, i \in \{1,2,3\}$ is incremented by a quantity defined by the parameter inc_i, an integer number between 0 and 255.

The *Evaporation* is the process of gradual disappearance of a substance, i.e. the level of the substances decreases over time. Let us define $\epsilon_i(v)$ as the function that computes the quantity of substance i to decrement from the substance level and is defined as

$$\epsilon_i(s) = s \cdot e_i^{(1)} + e_i^{(0)} \tag{7.1}$$

where $e_i^{(0)} \in \mathbb{R}^+$ is a constant evaporation quantity and $e_i^{(1)} \in \mathbb{R}, 0 \leq e_i^{(1)} \leq 1$ is the evaporation rate (e.g. a value of 0.1 means a 10% evaporation rate).

The evaporation function $evp_i(s)$, computing the level of substance s from time t to $t+1$, is thus defined as

$$evp_i(s) = \begin{cases} 0 & \text{if } \epsilon_i(s) > s \\ s - \epsilon_i(s) & \text{otherwise} \end{cases} \tag{7.2}$$

The *Diffusion* process simulates the diffusion of the substances through the cells. The body cell are disposed in a regular two-dimensional 4×4 square grid. We suppose that the cell $C_{x,y}$ is located on the grid at the position *i, j*, where $i \in \mathbb{N}$ and $j \in \mathbb{N}$. According to the von Neumann neighborhood (Gutowitz, 1991), a cell $C_{x,y}$ has the 4 neighbors $C_{x-1,y}$, $C_{x,y+1}$, $C_{x+1,y}$, $C_{x,y-1}$. Also the cells on the border have four neighbors: one or two of neighbors are *Edge Cells*.

For simplicity, we numbered the neighbors of a cell from 1 to 4, so for the cell $C_{x,y}$, N_1 is $C_{x-1,y}$, N_2 is $C_{x,y+1}$, N_3 is $C_{x+1,y}$, and N_4 is $C_{x,y-1}$. The substance level i of the neighbor cell n is indicated with $s_{n,i}$. The mean of the substance levels sn_i of the neighbors cell is computed as:

$$sn_i = \frac{\sum_{n=1}^{4} s_{n,i}}{c} \tag{7.3}$$

where c is the number of *connected* neighbors. A neighbor is considered connected if it is a Body Cell or if it is a Edge Cell connected to a neighborhood module.

The new value of each substance level s_1, s_2, s_3 is computed as:

$$new\ s_i = \frac{sn_i \cdot q + s_i \cdot (1-q)}{2} \tag{7.4}$$

where $q \in \mathbb{R}, 0 \leq q \leq 1$ is the sensitivity coefficient (i.e. if q is equals to 0, the new state of a cell is not influenced by the neighbors values, if it is equals to 0.5 the new values is a mean among the previous value of the cell and the neighbors value, if it is equals to 1, the new value does not depend on the previous value of the cell but only from the neighbors). The computed value are rounded to integer before the assignment of the value to the substance level.

Examples of the effect of the different parameters on the amount of substance are shown in Figures 7.9, 7.10 and 7.11.

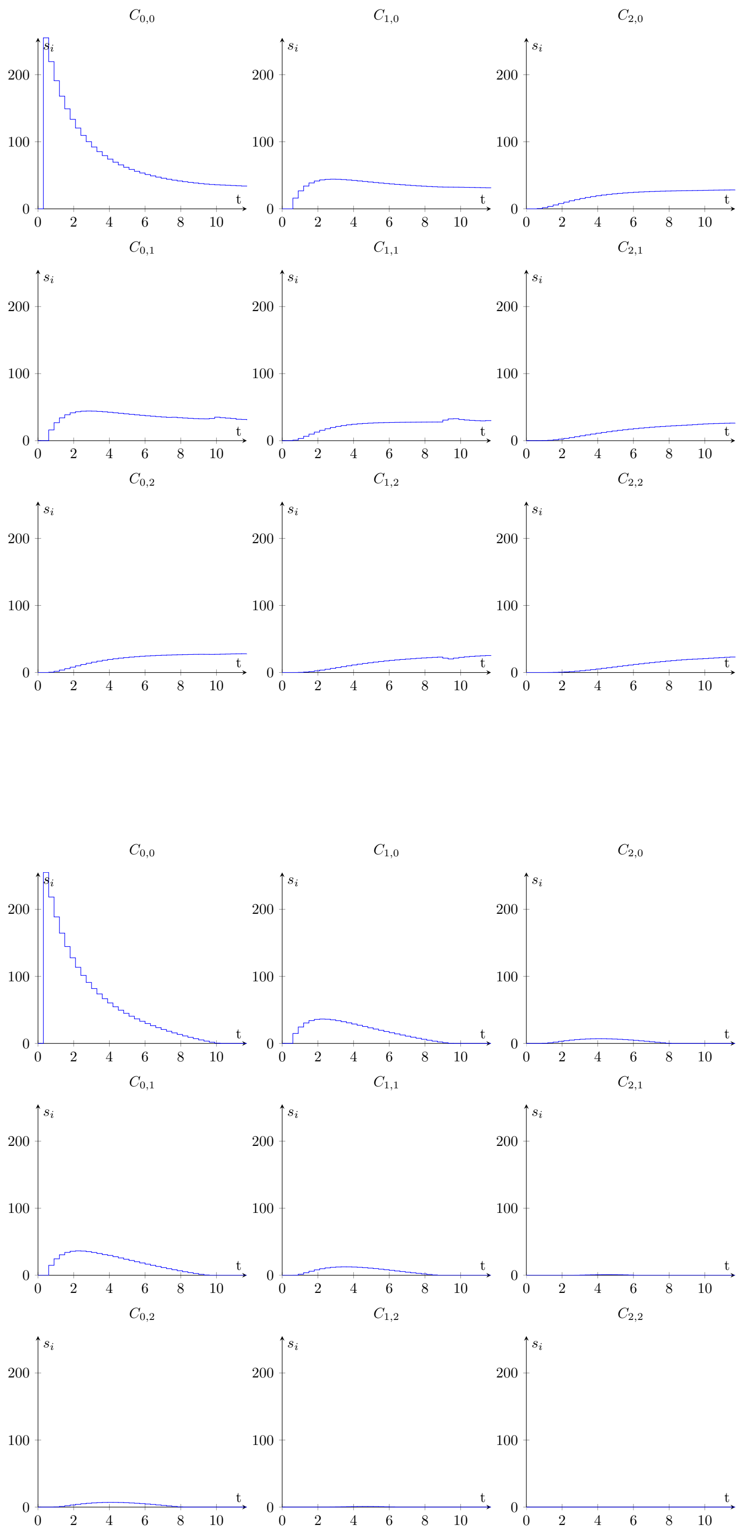


Figure 7.9: *Examples of dynamic evolution of substance level on 9 cells. The cell $C_{0,0}$ is stimulated after 0.3 s, the sensitivity parameter q is equal to 0.1, $inc_i = 255$. On the top without evaporation $e^{(0)} = 0$, $e^{(1)} = 0$, on the bottom with evaporation $e^{(0)} = 0.2$ $e^{(1)} = 0$.*

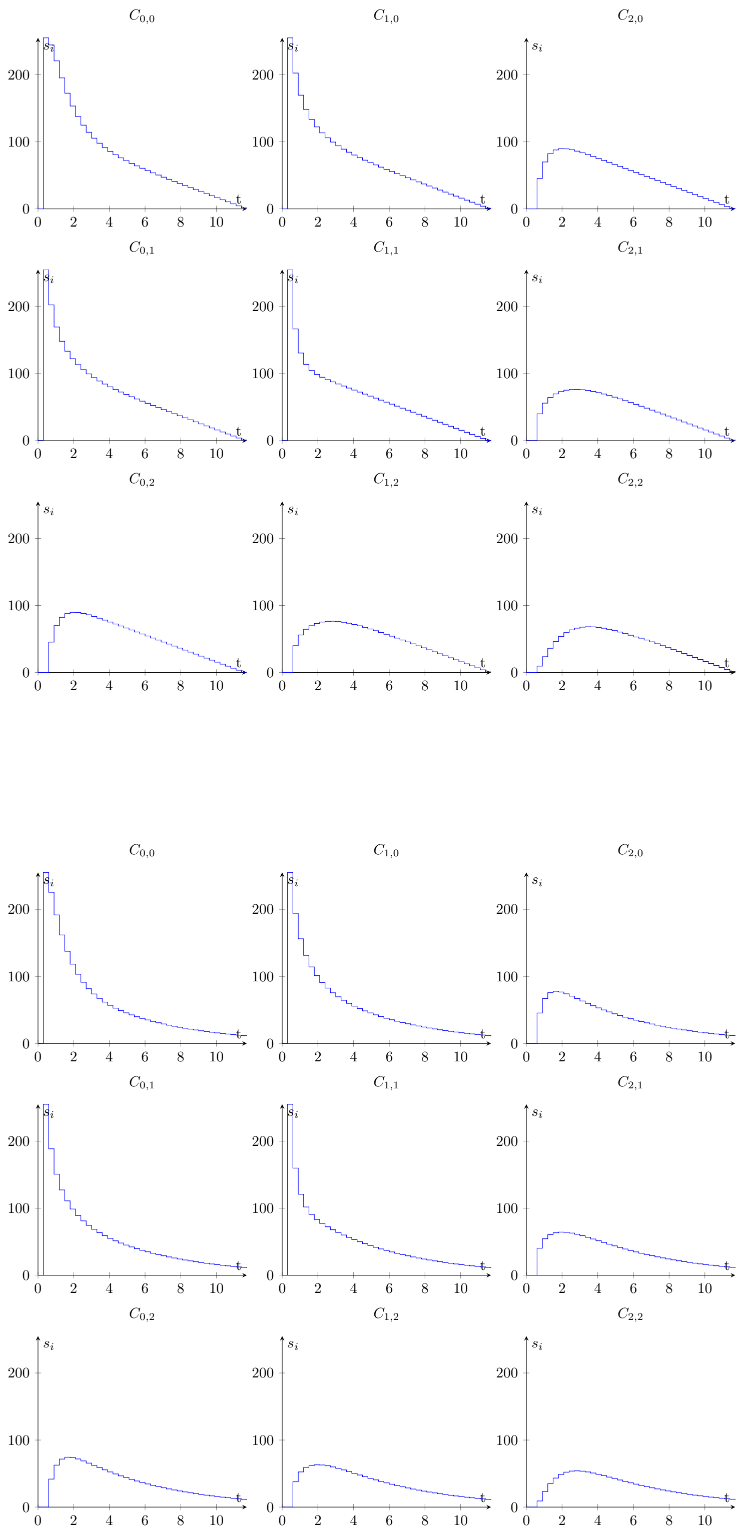


Figure 7.10: *Examples of dynamic evolution of substance level. The cells $C_{0,0}$,$C_{0,1}$,$C_{1,0}$ and $C_{1,1}$ are stimulated after 0.3 s, the sensitivity parameter q is equal to 0.3, $inc_i = 255$. On the top with constant evaporation $e^{(0)} = 0.5$, $e^{(1)} = 0$, on the bottom with proportional evaporation $e^{(0)} = 0$ $e^{(1)} = 0.01$.*

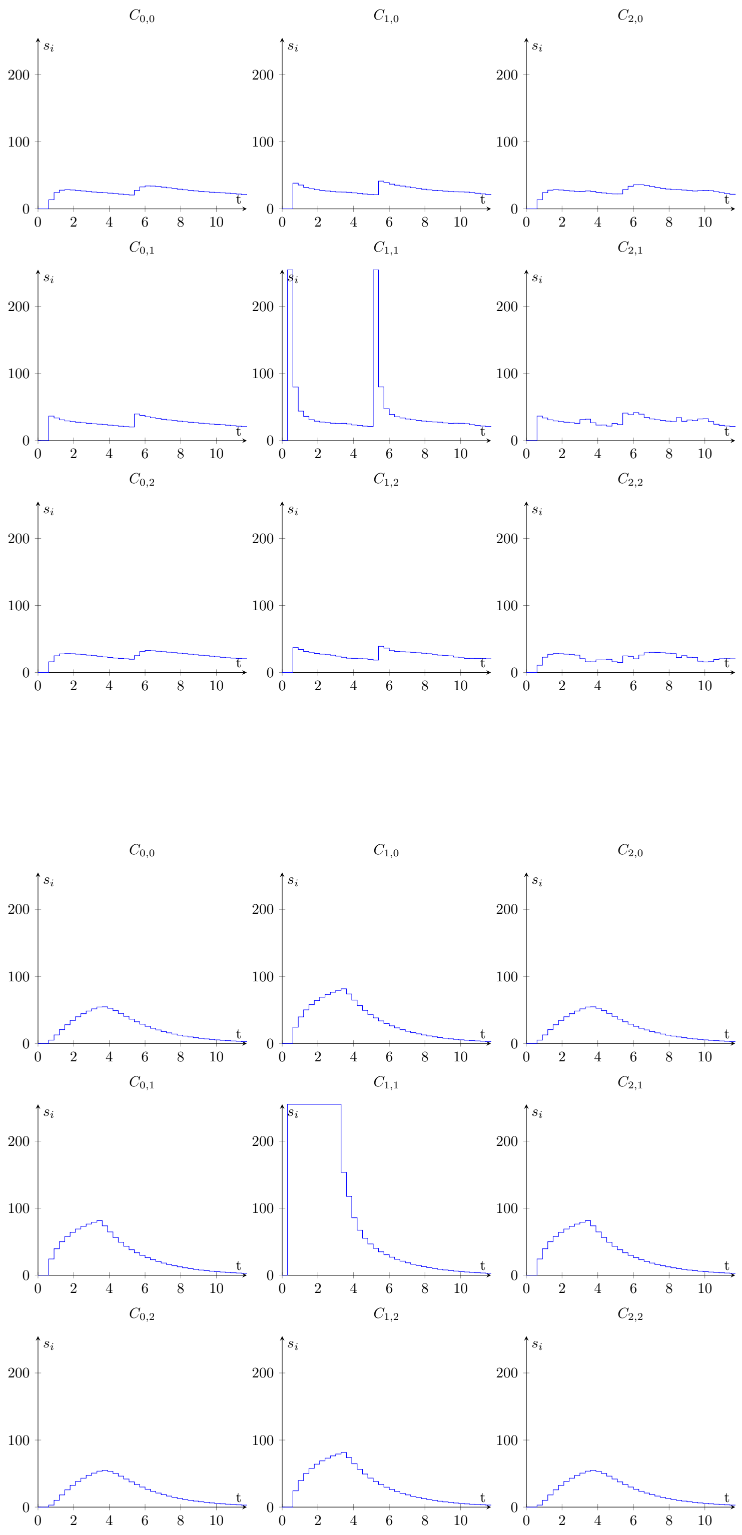


Figure 7.11: *Examples of dynamic evolution of substance level. On the top, the cell $C_{1,1}$ is stimulated two times (after 0.3 s and after 4.5), $q = 0.4$, $inc_i = 255$, $e^{(0)} = 0$, $e^{(1)} = 0.005$. On the bottom the same cell is stimulated for 3 s, $q = 0.2$, $inc_i = 255$, $e^{(0)} = 0$, $e^{(1)} = 0.02$.*

7.3.2 Actuator Cell

The actuators cells control the leds activities. Each actuator cell is connected to exactly one RGB led and one body cell. The actuator cell control the led according to the substance level of the Body cell. The RGB led has three independently controllable led: one red, one green and one blue. The led actuation is controlled by an intensity value between 0 and 255. The three intensity values, denoted with l_r, l_g, l_b, are computed as:

$$l_r = s_1 \cdot c_r \tag{7.5a}$$
$$l_g = s_2 \cdot c_g \tag{7.5b}$$
$$l_b = s_3 \cdot c_b \tag{7.5c}$$

where s_1, s_2, s_3 are the substance amounts and c_r, c_g, c_b are the three components of the color parameter.

7.3.3 Edge Cell

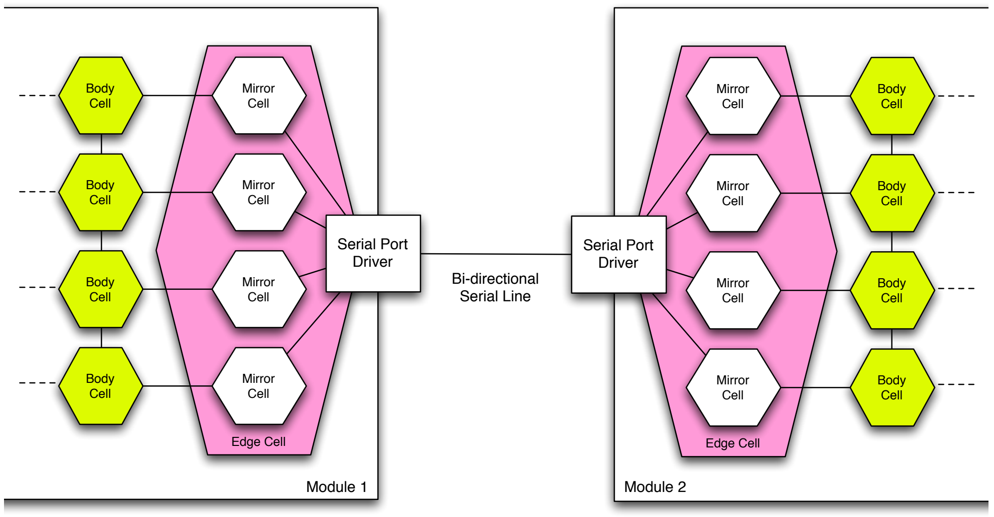


Figure 7.12: *Schematization of the interactions between the edge cells and the body cells.*

The aim of the Edge Cells is to communicate with the other modules. Internally, each edge cells acts has 4 "mirror" cells connected to the near body cell. A schematization of the mirror cell is shown in Figure 7.12. For the body cells, the mirror cells appear identical to the other body cell, i.e. each mirror cell exposes three substances levels. The mirror cells act as a remote copies of the body cell connected on the other side of the serial connection. Through the mirror cell, the body cells values are send over the serial connection the other module. The activity of the edge cells can be reassumed in two tasks:

- Transmission task - During the transmission task, the values of the body cells connected to the edge cell are serialized in a message. The message is send over the serial line by the serial driver.
- Reception task - A message received through the serial driver is deserialized and the values of the body cells of the other module are copied into the mirror cell.

7.3.4 Command Shell Cell

The developed prototype provides a command-line shell for interacting with the Adaptive Lighting system. The users are able to interact (e.g. for debugging and configuring) with the cells through a serial terminal. The command shell is implemented as a particular cell that receives the commands from the serial line, executes the command, and sends the response to the terminal. The serial line used by the shell is a one wire serial connection shared by all the cell. This means that a command sent over the serial connection is received to all the modules belonging to the same installation.

In order to send command to specific modules, each modules is characterized by a 16-bit address. When a command is preceded by a module address (represented as a hexadecimal number), all the modules with a different address ignore the command. For example, the following command

```
ff ev0 10
```

is interpreted only by the module with address 255 (FF in in hexadecimal) and discarded by the other modules. All the lines starting with the # character are discarded. In order to avoid that a response from a module is interpreted as a command from another module (because the modules are connected through a one wire serial connection), all the responses generated by the modules start with the # character. After the # character, each response contains the address of the module. For example, the following message

```
#10 r 64 g 128 b 64
```

was generated by the module with address 16 (10 in hexadecimal)).

The following is the list of the available commands:

help Display helpful information about builtin commands. This comand does not accept any parameters. Example:

```
help
#10 color
```

```
#10 help
#10 ev0
#10 ev1
#10 inc
#10 sens
#10 sn
#10 edge
#10 ps
#10 prox
```

color Display or set the components (c_r, c_g, c_b) of the color parameter. This values are composed with the substance levels in order to control the RGB leds. The color components are represented as integer number between 0 and 255. Example of displaying the color parameter:

```
color
#10 r 64 g 128 b 64
```

Example of setting a new value for the color parameter:

```
color 255 0 0
#10 r 255 g 0 b 0
```

ev0 Display or set the $e_i^{(0)}$ parameters (constant evaporation). There is a separate value of the parameter for each substance. If the command is invoked without parameters, the current values of the parameter are displayed. If it is invoked with one parameter, the same new value is assigned to each $e_i^{(0)}$ parameters. To set a different value for each substance, invoke the command with 3 arguments. The parameters range is an integer number between 0 and 255. Example of displaying the evaporation $e_i^{(0)}$:

```
ev0
#10     16     16     16
```

Example of settings $e_1^{(0)}$ to 16, $e_2^{(0)}$ to 20, and $e_3^{(0)}$ to 23:

```
ev0 16 20 23
#10     16     20     23
```

ev1 Display or set the $e_i^{(1)}$ parameters (proportional evaporation). There is a separate value of the parameter for each substance. If the command is invoked without parameters, the current values of the parameter are displayed. If it is invoked with one parameter, the same new value is assigned to each $e_i^{(1)}$ parameters. To set a different value for each substance, invoke the command with 3 arguments. The parameters range is an integer number between 0 and 255. Example of displaying the evaporation $e_i^{(1)}$:

```
ev1
#10     0      0      0
```

Example of settings $e_{1..3}^{(1)}$ to 32:

```
ev1 32
#10     32     32     32
```

inc Display or set the inc_i parameters, the substance increments in response to sensors stimulus. If the command is invoked without parameters, the current values of the parameter are displayed. If it is invoked with one parameter, the same new value of the increment is assigned to each inc_i parameters. To set a different vale for each channel, invoke the command with 3 arguments. Example of displaying inc_i:

```
inc
#10     40     40     40
```

Example of settings $inc_{1..3}$ to 32:

```
inc 32 32 32
#10     32     32     32
```

sens Display or set the q parameter (sensitivity). The value of the parameter is an integer number between 0 and 255. The value 0 corresponds to $q = 0$, the value 255 corresponds to $q = 1$. Example of displaying the parameter:

```
sens
#10     40
```

Example of settings the sensitivity parameter to 51 ($q = 0.2$):

```
sens    51
#10     51
```

sn Display or set the *substance intensities* of each body cell of the module. Example of displaying the substance intensities:

```
sn
#10 0x0000  31219  31219  31219
#10 0x0001  31151  31151  31151
```

```
#10 0x0002  31142  31142  31142
#10 0x0003  31103  31103  31103
#10 0x0004  31151  31151  31151
#10 0x0005  31107  31107  31107
#10 0x0006  31102  31102  31102
#10 0x0007  31060  31060  31060
#10 0x0008  31142  31142  31142
#10 0x0009  31102  31102  31102
#10 0x000a  31097  31097  31097
#10 0x000b  31052  31052  31052
#10 0x000c  31103  31103  31103
#10 0x000d  31060  31060  31060
#10 0x000e  31052  31052  31052
#10 0x000f  30975  30975  30975
```

Example of settings the substance intensities to 128 for all the substances and all the cells:

```
sn 128
```

edge Display the information about the edge cells. For each edge cells, the command reports the edge direction, the serial line, transmission and reception status, and the state of internal mirror cells. Example:

```
edge
#10 direction: 1
#10 line: 3
#10 tx/rx: rt
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 direction: 2
#10 line: 4
#10 tx/rx: rt
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 direction: 3
#10 line: 1
#10 tx/rx: rt
```

```
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 direction: 0
#10 line: 2
#10 tx/rx: rt
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
#10 0x00ff      0      0      0
```

ps The *ps* command displays lines containing information about all the cells of the module. It also displays the total number of cells in the modules and the amount of free memory. Example:

```
ps
#10   n update   state    s    mem
#10 -----------------------------
#10   0 0x404a80 0x000000 m      0
#10   1 0x401fff 0x409374 m     76
#10   2 0x401c4e 0x4093c0 m      8
#10   3 0x401fff 0x4093c8 m     76
#10   4 0x401c4e 0x409414 m      8
#10   5 0x401fff 0x40941c m     76
#10   6 0x401c4e 0x409468 m      8
#10   7 0x401fff 0x409470 m     76
#10   8 0x401c4e 0x4094bc m      8
#10   9 0x401fff 0x4094c4 m     76
#10  10 0x401c4e 0x409510 m      8
#10  11 0x401fff 0x409518 m     76
#10  12 0x401c4e 0x409564 m      8
#10  13 0x401fff 0x40956c m     76
#10  14 0x401c4e 0x4095b8 m      8
#10  15 0x401fff 0x4095c0 m     76
#10  16 0x401c4e 0x40960c m      8
#10  17 0x401fff 0x409614 m     76
#10  18 0x401c4e 0x409660 m      8
#10  19 0x401fff 0x409668 m     76
#10  20 0x401c4e 0x4096b4 m      8
#10  21 0x401fff 0x4096bc m     76
```

```
#10  22 0x401c4e 0x409708 m      8
#10  23 0x401fff 0x409710 m     76
#10  24 0x401c4e 0x40975c m      8
#10  25 0x401fff 0x409764 m     76
#10  26 0x401c4e 0x4097b0 m      8
#10  27 0x401fff 0x4097b8 m     76
#10  28 0x401c4e 0x409804 m      8
#10  29 0x401fff 0x40980c m     76
#10  30 0x401c4e 0x409858 m      8
#10  31 0x401fff 0x409860 m     76
#10  32 0x401c4e 0x4098ac m      8
#10  33 0x405775 0x409bb4 m     40
#10  34 0x405775 0x409d0c m     40
#10  35 0x405775 0x409e64 m     40
#10  36 0x405775 0x409fbc m     40
#10  37 0x403d35 0x000000 m      0
#10  38 0x401380 0x000000 m      0
#10  39 0x4013c1 0x000000 d      0
#10  40 0x404df0 0x40a114 m      8
#10 number of cells: 41
#10 free memory: 852
```

prox Display the status of the proximity sensors. Example:

```
prox
#10  0 0
#10  1 0
#10  2 0
#10  3 0
#10  4 0
#10  5 0
#10  6 0
#10  7 0
#10  8 0
#10  9 0
#10 10 0
#10 11 0
#10 12 0
#10 13 0
#10 14 0
```

```
#10 15 0
#10 0x0000
```

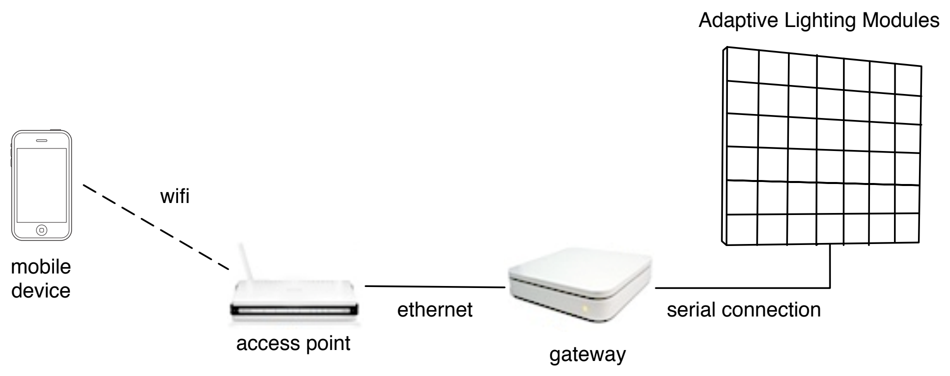


Figure 7.13: *Connection between the configuration interface running on an iPhone and an adaptive lighting installation.*

7.4 The Configuration Interface

In this section we present a prototypal application allows to configure a modular adaptive lighting system from an iPhone or an iPod Touch. We chosen these devices because they are multi-touch handheld devices with a large display covering most of the top surface and they have a wifi connection. In fact, the configuration interface interacts with the adaptive lighting installation though wifi, as depicted in Figure 7.13. Moreover the proposed application can be easy ported to different hardwares, both mobile devices and dedicated embedded system.

The main menu, shown in Figure 7.14, is the first menu displayed when the program is opened. Starting from the main menu, the user has access to various configuration.

The *stimulus response configuration interface* allows the user to set the inc_i parameters, i.e. the amounts of virtual substance to be added to the cells when a sensor is stimulated. The user can individually set the parameters using slider bars.

The *diffusion control interface* (Figure 7.15) allows the user to set the sensitivity coefficient using a slider bar. The largest part of the screen is devoted to the simulation of the effect of the different values of the parameter. The multi-touch screen allows the user to stimulated the cells touching them.

The evaporation configuration interface, shown in Figure 7.16, allows to set the parameters regarding the constant and proportional evaporation (i.e. $e^{(0)}_{1..3}$ and $e^{(1)}_{1..3}$). A graph in the interfaces allows to visualize the effect of the different configurations.

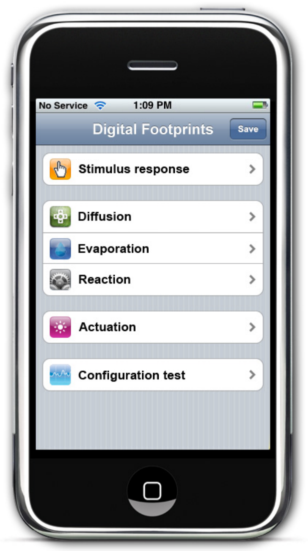
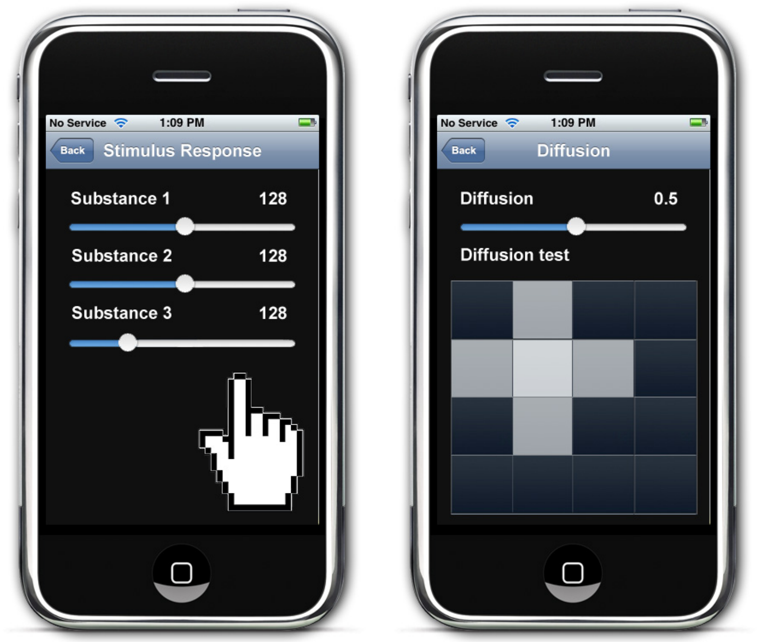


Figure 7.14: *Screenshots of the main menu of the configuration application.*

The actuations library, shown in Figure 7.17, allows to enable different actuation cells. The behavior of the default actuator cells is described in Section 7.3.2. Other behaviors can be defined through MDCA cell definitions. This interfaces allows to chose the desire behavior and to configure them. In the proposed screenshot is shown the only configurable parameter of the default behavior: the color. Moreover different actuator cells can have others user-settable parameters.

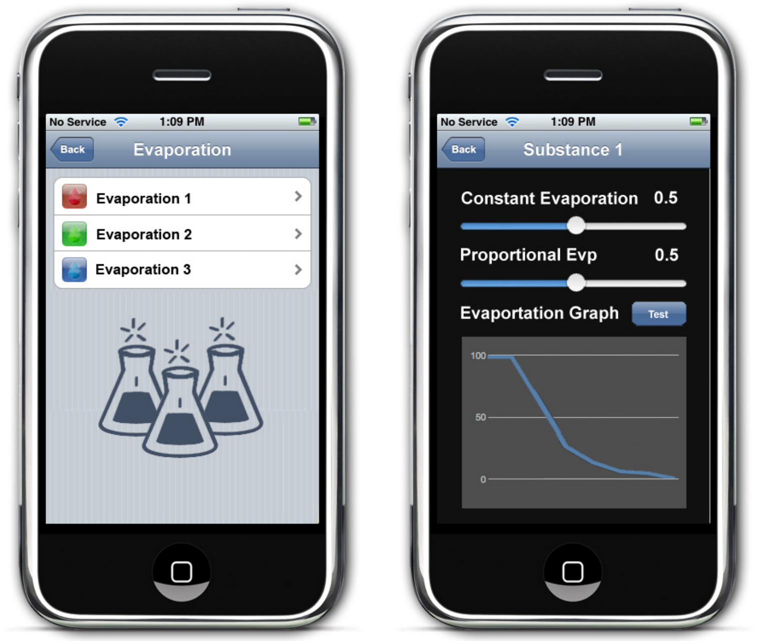
The reaction library, shown in Figure 7.18, allows to enable different reaction cells. A reaction cells is a cell that "reacts" to the presence (or absence) of a substance, converting one or more substances into other substance. In the default configuration of the modular lighting system, reaction cells are not present. An example of cellular automata for reaction-diffusion systems is presented in Weimar (1997a). As for the actuator cells, the reaction cells behaviors can be defined through MDCA cell definitions. Each kind of reaction cell can have several user-settable parameters.

The application can simulate an installation composed of several lighting modules, as shown in Figure 7.18. The simulation take care of the parameters configured throughs the other forms and display the dynamic behavior of the cells. The user can stimulate the cells touching the screen, in order to test how they react to the stimuli. Finally the configuration can be saved on the devices and deployed in a real installation.



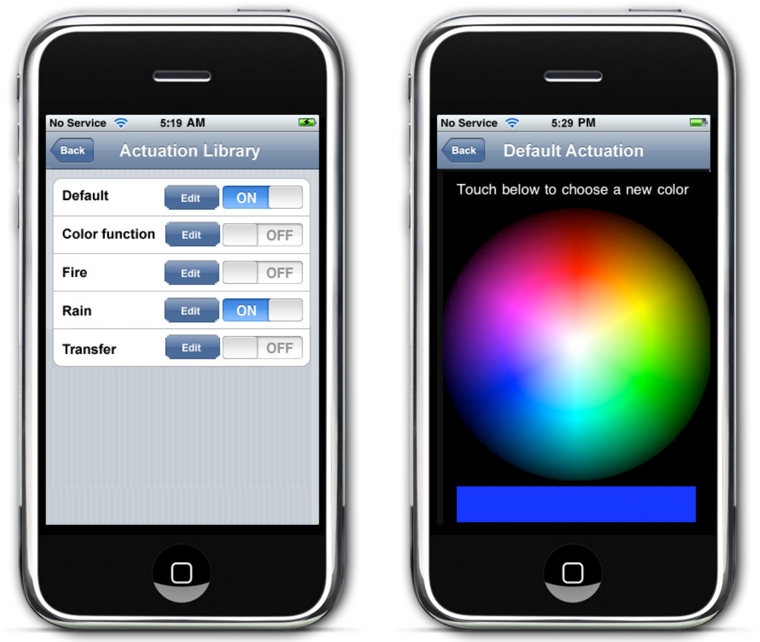
Substance 1 128
Substance 2 128
Substance 3 128
Diffusion 0.5

Figure 7.15: *Screenshots of the stimulus response configuration interface (on the left) the diffusion control interface (on the right).*



Constant Evaporation 0.5
Proportional Evp 0.5

Figure 7.16: *Screenshots of the evaporation configuration interface.*


No Service 5:19 AM

Back Actuation Library

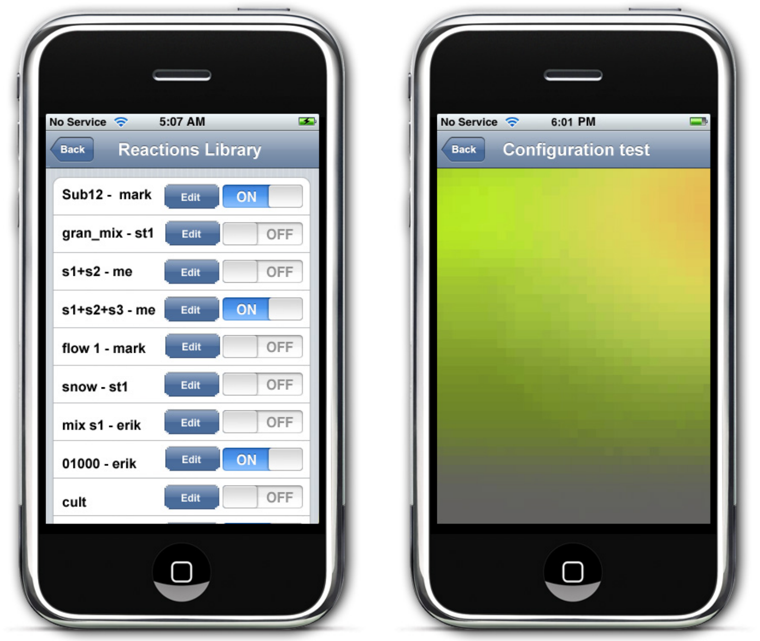
Default	Edit	ON
Color function	Edit	OFF
Fire	Edit	OFF
Rain	Edit	ON
Transfer	Edit	OFF

No Service 5:29 PM

Back Default Actuation

Touch below to choose a new color

Figure 7.17: *Screenshots of the actuations library (on the left) and configuration of the default actuation scheme (on the right).*


No Service 5:07 AM

Back Reactions Library

Sub12 - mark	Edit	ON
gran_mix - st1	Edit	OFF
s1+s2 - me	Edit	OFF
s1+s2+s3 - me	Edit	ON
flow 1 - mark	Edit	OFF
snow - st1	Edit	OFF
mix s1 - erik	Edit	OFF
01000 - erik	Edit	ON
cult	Edit	OFF

No Service 6:01 PM

Back Configuration test

Figure 7.18: *Screenshots of the reactions library (on the left) and configuration simulation (on the right).*

7.5 Considerations

This chapter introduced a prototype of a MDCA-based modular adaptive lighting system. The proposed system is currently being developed. An hardware prototype has been developed by Egicon. This prototype is important both for testing the validity of the proposed model for the realization of real adaptive lighting installation and to demonstrate the concept of modular adaptive lighting to the designers and the potential users.

Future work is intended to identify and build the infrastructure required for enabling designers and final users to independently create and configure lighting installation. One of such tools is the proposed configuration interfaces for the iPhone and iPod Touch. Such interface needs to be extended allowing to easy configure the parameters and the shape of the installation. However the proposed interfaces shown that controlling adaptive lighting installation can be easy also for non-expert users.

The whole is more than the sum of its parts.

Aristotle, Metaphysica 10f-1045a

8 Conclusions and Future Developments

8.1 Conclusion

THE starting point of this work was the following question: is it possible to control in a comfortable way Adaptive Lighting systems with a cellular distributed control system in order to take advantage of the cellular systems' qualities?

In order to address this question, several specific objectives has been defined. This section highlights the accomplished objectives.

OBJECTIVE 1 *To define a cellular automata model suitable for distributed control and in particular for the control of Adaptive Lighting installations.*

The proposed model, called Multilayered Dissipative Cellular Automata (MDCA), is an *Asynchronous* (i.e. the cells can be updated according to several update schemes, both synchronous and asynchronous), *Heterogeneous* (i.e cells are heterogeneous, in terms of space of the states and transition rule), *Multilayered* (i.e. cellular space is a hierarchical structure), and *Dissipative* (i.e the dynamic behavior of the automata is influenced by the external environment and influences the external environment) Cellular Automata.

Such features are useful for designing systems composed of several distributed interacting components. MDCA can be used both to simulate the behavior of such distributed systems and to control the real Adaptive Lighting installations. Adaptive Lighting is in fact a challenging Ambient Intelligent application and a positive result in the above direction represent a starting point for a more general application even in other scenarios.

OBJECTIVE 2 *To investigate the effect of the asynchronous nature of cellular distributed control systems, in order to understand the problematics deriving from the adoption of an asynchronous model that is much more adequate to distributed systems than a synchronous one.*

We analyzed several cellular automata update schemes presented in literature (e.g. Random order, Fixed Random Order, Interlaced order, Synchronous Scheme, Random Independent, Random Order, Cyclic, Clocked) and outlined a tentative classification of such schemes. To the best of our knowledge, this is the first update schemes' classification.

In order to study the effects of the different update schemes, we introduced a class of very simple CA, called One Neighbor Binary Cellular Automata (1nCA). We presented the effects of several update schemes on this class and classified the automata according to how much the dynamic evolution is influence by the update scheme.

OBJECTIVE 3 *To build a prototype of a real Adaptive Lighting installation in order to verify it the proposed model is suitable.*

During the Indianapolis Project we developed a first tentative step toward a cellular automata-based model for Adaptive Lighting. During the project, we worked with the designer of the Acconci studio in order to deeply understand the light designers' requirements. Our aim was to facilitate the user in designing the dynamic behavior of a lights installation by supporting the envisioning of the effects of a given configuration for the transition rule guiding lights; therefore we created an ad-hoc tool, comprising both a simulation environment and a graphical parameters configurator. This tool support the specification of the values for some of the parameters of the transition rule, affecting the global behavior of the overall system. The integrated simulation helps understanding how the changes of the single parameters influence the overall behavior of the illumination facility: every changed parameter is immediately used in the transition rule of every cell. This experience with the prototype lead us to conclude that it is feasible to control Adaptive Lighting installation through the proposed CA model. This work was published in Bandini et al. (2008) and Bandini et al. (2009).

OBJECTIVE 4 *To define a programming language suitable for creating control systems with the proposed model.*

In order to enable the users to define by theirself the cells' behavior, we define a textual and a visual programming languages. The textual programming language, MDCA Language is a cellular automata programming language that allows the user to create automata creating new cells and combining existing cells. MDCA Visual Programming Editor is a graphical programming environment for the MDCA Language, but using this editor, the textural language is hidden to the user. A visual programming language was required because visual programming languages are popular in communities of designer and artists.

OBJECTIVE 5 *To create a cellular execution environment suitable for the class of microcontrollers typically used in distributed control systems.*

In order to effectively the MDCA model and programming language for the control of real Adaptive Lighting installation, we developed a run-time environment for the MDCA cells suitable for the class of microcontrollers typically used in these systems. The runtime environment offers some functionalities of a "traditional" operating system (e.g. as hardware abstraction, scheduling, memory management, I/O). In fact, it can be considered simple operating system designed to run on 8-bit and 16-bit microcontroller. The environment was successfully tested on different microcontrollers.

OBJECTIVE 6 *To design an Adaptive Lighting system module based on the proposed model suitable for different adaptive lighting installations.*

In 2009 CSAI (Complex System and Artificial Intelligence Research Centre) started a collaboration with Egicon, an italian electronic engineering company, for designing and developing an modular lighting system based on MDCA. The aim of the collaboration between CSAI an Egicon is the development of a prototype of a hardware platform suitable for a modular Adaptive Lighting.

In collaboration with Egicon, we had developed a Modular Adaptive Lighting System composed of several independent modules, equipped with proximity sensors, and RGB leds guided by the proposed model. A module can be used alone or together with other modules. The modules are square shaped and they can be combine like "bricks" in order to create a larger lighting system.

Also, we had developed a prototypal application allows to configure a modular adaptive lighting system from iPhone and iPod Touch devices.

8.2 Future Developments

The work that has been presented in this thesis can be further continued in other directions. Let us describe some of them:

- Further investigation of the effect of asynchronicity. In addition, we expect that the model can also be useful for further studying the emergence of collective behavior of asynchronous cellular automata.
- To adopt the MDCA approach in other Ambient Intelligence scenarios, e.g.:
  - *Dynamic Wayfinding* To develop a dynamic wayfinding system, equipped with sensors and actuators, that provides to the pedestrians adaptive indications for reaching the building's exit according to the current state of the path (e.g. crowded or dangerous areas).
  - *Demain Control Ventilation* Demand-Control Ventilation (DCV) is a method of controlling the ventilation of a space based on a continuously monitoring of the CO_2 level, in the same way the thermostats regulate the amount of cooling or heating supplied to a building space. MDCA can be adopted to this scenario to create intelligent ventilation devices able to interact with the other facilities of the building (e.g. the access control system). Each node is responsible to maintain the required air quality in a fixed region of the space but could communicate with the neighborhood MDCA nodes.

Bibliography

Aiello, M. and Dustdar, S. (2008). Are our homes ready for services? a domotic infrastructure based on the web service stack. *Pervasive and Mobile Computing*, 4(4):506 – 525.

Albin, P. S. (1975). *The Analysis of Complex Socioeconomic Systems*. D. C. Health and Company/Lexington Books, Lexington, MA.

Albin, P. S. and Foley, D. K. (1999). Barriers and bounds to rationality: Essays on economic complexity and dynamics in interactive systems. *J. Artificial Societies and Social Simulation*, 2(4).

Aldana, M., Coppersmith, S., and Kadanoff, L. P. (2003). *Boolean dynamics with random couplings*, pages 23–89. Springer-Verlag.

Allen, B., van Berlo, A., Ekberg, J., Fellbaum, K., Hampicke, M., and Willems, C. (2001). *Design Guidelines on Smart Homes*. COST 219bis guidebook.

Alonso-Sanz, R. (2003). Reversible cellular automata with memory: patterns starting with a single site seed. *Physica D*, 175(1–2):1–30.

Alonso-Sanz, R. (2004). One-dimensional, r=2 cellular automata with memory. *International Journal of Bifurcation and Chaos*, 14(9):3217–3248.

Alonso-Sanz, R. (2006). The beehive cellular automaton with memory. *Journal of Cellular Automata*, 1(3):195–211.

Alonso-Sanz, R. (2007). A structurally dynamic cellular automaton with memory. *Chaos, Solitons & Fractals*, 32(4):1285–1295.

Alonso-Sanz, R. and Cardenas, J. P. (2008). On the effect of memory in one-dimensional k=4 automata on networks. *Physica D*, 237(23):3099–3108.

Alonso-Sanz, R. and Martín, M. (2002). One-dimensional cellular automata with memory: patterns starting with a single site seed. *International Journal of Bifurcation and Chaos*, 12(1):205–226.

Alonso-Sanz, R. and Martin, M. (2003). Elementary cellular automata with memory. *Complex Systems*, 14(2):99–126.

Araya, A. A. (1995). Questioning ubiquitous computing. In *CSC '95: Proceedings of the 1995 ACM 23rd annual conference on Computer science*, pages 230–237, New York, NY, USA. ACM.

Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Book, New York, NY.

Balzer, R. (1967). An 8-state minimal time solution to the firing squad synchronization problem,. *Information and Control*, 10(1):22 – 42.

Bandini, S., Bonomi, A., and Vizzari, G. (2009). Simulation supporting the design of self-organizing ambient intelligent systems. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2082–2086, New York, NY, USA. ACM.

Bandini, S., Bonomi, A., Vizzari, G., Acconci, V., DeGraaf, N., Podborseck, J., and Clar, J. (2008). A ca-based self-organized illumination facility. In Brueckner, S. A., Robertson, P., and Bellur, U., editors, *SASO*, pages 485–486. IEEE Computer Society.

Bandini, S., Cattaneo, G., and Tarantello, G. (1992). Distributed ai models for percolation. In *Intelligent Scientific Computation*, AAAI Fall Symposium Series, Cambridge (MA).

Bandini, S., Manzoni, S., and Vizzari, G. (2006). Towards a platform for multilayered multi agent situated system based simulations: Focusing on field diffusion. *Applied Artificial Intelligence*, 20(4–5):327–351.

Bandini, S. and Mauri, G. (1999). Multilayered cellular automata. *Theor. Comput. Sci.*, 217(1):99–113.

Bandini, S. and Pavesi, G. (2002). Simulation of vegetable populations dynamics based on cellular automata. In *ACRI '01: Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*, pages 202–209, London, UK. Springer-Verlag.

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, ISBN 0-12-091152-3. chapter 25.

Binder, P. (1993). A phase diagram for elementary cellular automata. *Complex Systems*, 7:241–247.

Binder, P. (1994). Parametric ordering of complex systems. *Physical Review E*, 49(3):2023–2025.

Blue, V. J. and Adler, J. L. (1998). Emergent fundamental pedestrian flows from cellular automata microsimulation. *Journal of the Transportation Research Board*.

Blue, V. J. and Adler, J. L. (1999a). Bi–directional emergent fundamental pedestrian flows from cellular automata microsimulation. In *14th international symposiumon transportation and traffic theory*, pages 235–254.

Blue, V. J. and Adler, J. L. (1999b). Cellular automata micro-simulation of bi–directional pedestrian flows. *Journal of the Transportation Research Board*, pages 135–141.

Blue, V. J. and Adler, J. L. (2000). Cellular automata model of emergent collective bi–directional pedestrian dynamics. In *Artificial Life VII, The Seventh International Conference on the Simulation and Synthesis of Living Systems*.

Blue, V. J. and Adler, J. L. (2001). Cellular automata microsimulation for modeling bidirectional pedestrian walkways. *Trasportation Research Part B*.

Boer, R. J. D. and Hogeweg, P. (1992). Growth and recruitment in the immune network. In Perelson, A. F. and Weisbuch, G., editors, *Theoretical and Experimental Insights into Immunology*, volume 66, pages 223–247. Springer Verlag, New York.

Boon, J., Dab, D., Kapral, R., and Lawniczak, A. (1995). Lattice gas automata for reactive systems. *Physics Reports*, 273:55–147.

Burks, A., editor (1970). *Essays on Cellular Automata*. University of Illinois Press, Urbana.

Burstedde, C., Kirchner, A., Klauck, K., Schadschneider, A., and Zittartz, J. (2002). Cellular automaton approach to pedestrian dynamics - applications. *Pedestrian and Evacuation Dynamics*, pages 87–97.

Burstedde, C., Klauck, K., Schadschneider, A., and Zittartz, J. (2001). Simulation of pedestrian dynamics using a 2–dimensional cellular automata. *Physica*.

Buvel, R. and Ingerson, T. (1984). Structure in asynchronous cellular automata. *Physica D*, 1:59–68.

Campari, E. G., Levi, G., and Maniezzo, V. (2004). Cellular automata and roundabout traffic simulation. In Sloot et al. (2004), pages 202–210.

Cerdá, J., Gironés, R. G., Martínez, J. D., and Sebastia, A. (2005). A tool for implementing and exploring sbm models: Universal 1d invertible cellular automata. In Mira, J. and Álvarez, J. R., editors, *IWINAC (1)*, volume 3561 of *Lecture Notes in Computer Science*, pages 279–289. Springer.

Chen, H., Chen, S., Doolen, G., and Lee, Y. C. (1988). Simple lattice gas models for waves. *Complex Systems*, 2(3):259–267.

Chopard, B. and Droz, M. (1998). *Cellular Automata Modeling of Physical Systems*. Cambridge University Press.

Chopard, B., Luthi, P. O., and Queloz, P. A. (1996). Cellular automata model of car traffic in two-dimensional street networks. *Journal of Physics A: Mathematical and General*, 29:2325–2336.

Chopard, B., Luthi, P. O., and Wagen, J.-F. (1997). Lattice boltzmann method for wave propagation in urban microcells. *Microwaves, Antennas and Propagation, IEE Proceedings*, 144(4):251–255.

Cole, J. B., Krutar, R. A., Creamer, D. B., and Numrich, S. K. (1993). A cellular automation methodology for solving the wave equation. In *ICS '93: Proceedings of the 7th international conference on Supercomputing*, pages 348–356, New York, NY, USA. ACM.

Cornforth, D., Green, D. G., and Newth, D. (2005). Ordered asynchronous processes in multi-agent systems. *Physica D: Nonlinear Phenomena*, 204(1-2):70 – 82.

Dab, D., Lawniczak, A., Boon, J.-P., and Kapral, R. (1990). Cellular-automaton model for reactive systems. *Phys. Rev. Lett.*, 64(20):2462–2465.

De Oliveira, N., Oxley, N., Petry, M., and Archer, M. (1994). *Installation Art*. Smithsonian Institution Press.

Deutsch, A. and Dormann, S. (2008). *Cellular Automaton Modeling of Biological Pattern Formation*. Springer.

Edmonds, E., Turner, G., and Candy, L. (2004). Approaches to interactive art systems. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 113–117, New York, NY, USA. ACM.

Edmonds, E. A., Weakley, A., Candy, L., Fell, M., Knott, R., and Pauletto, S. (2005). The studio as laboratory: Combining creative practice and digital technology research. *Int'l Journal of Human Computer Studies*, 63:4–5.

Ehleringer, J. and Forseth, I. (1980). Solar tracking by plants. *Science*, 210(4474):1094–1098.

Elliott, C. and Hudak, P. (1997). Functional reactive animation. In *International Conference on Functional Programming*, Amsterdam, The Netherlands. ACM Press, New York.

Fatès, N. (2003). Experimental study of elementary cellular automata dynamics using the density parameter. In Morvan, M. and Rémila, é., editors, *Discrete Models for Complex Systems, DMCS'03*, volume AB of *DMTCS Proceedings*, pages 155–166. Discrete Mathematics and Theoretical Computer Science.

Friedewald, M. (2005). Safeguards in a world of ambient intelligence. In Hutter, D. and Ullmann, M., editors, *SPC*, volume 3450 of *Lecture Notes in Computer Science*, pages 63–69. Springer.

Frisch, U., Hasslacher, B., and Pomeau, Y. (1986). Lattice-gas automata for the Navier-Stokes equation. *Physical Review Letters*, 56:1505–1508.

Ganguly, N., Das, A., Maji, P., Sikdar, B. K., and Chaudhuri, P. P. (2001). Evolving cellular automata based associative memory for pattern recognition. In *HiPC '01: Proceedings of the 8th International Conference on High Performance Computing*, pages 115–124, London, UK. Springer-Verlag.

Ganguly, N., Maji, P., Das, A., Sikdar, B. K., and Chaudhuri, P. P. (2002). Characterization of non-linear cellular automata model for pattern recognition. In *AFSS '02: Proceedings of the 2002 AFSS International Conference on Fuzzy Systems. Calcutta*, pages 214–220, London, UK. Springer-Verlag.

Gardner, M. (1970). The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123.

Gardner, M. (1983). *Wheels, Life, and Other Mathematical Amusements*. W. H. Freeman and Company, New York. ISBN 0-7167-1589-9.

Gershenson, C., Broekaert, J., Aerts, D., and Apostel, C. L. (2003). Contextual random boolean networks. In *In Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems (ALife IX*, pages 1–8. MIT Press.

Golès, E. and Martinez, S. (1990). *Neural and Automata Networks: Dynamical Behavior and Applications*. Kluwer Academic Publishers. ISBN 0-792-30632-5.

Goodell, H., Kuhn, S., Maulsby, D., and Traynor, C. (1999). End user programming/informal programming. *ACM SIGCHI Bulletin*, 31(4):17–21.

Gutowitz, H. (1991). *Cellular Automata: Theory and Experiment*. MIT Press/Bradford Books, Cambridge Mass. ISBN 0-262-57086-6.

Gutowitz, H., Victor, J. D., and Knight, B. W. (1987). Local structure theory for cellular automata. *Physica D*, 28:18–48.

Hannington, A. and Reed, K. (2002). Towards a taxonomy for guiding multimedia application development. *Asia-Pacific Software Engineering Conference*, page 97.

Hardy, J., de Pazzis, O., and Pomeau, Y. (1976). Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *Physical Review A*, 13:1949–1961.

Harvey, I. and Bossomaier, T. (1997). Time out of joint: Attractors in asynchronous random boolean networks. In *Proceedings of the Fourth European Conference on Artificial Life (ECAL97*, pages 67–75. MIT Press.

He, M., Pan, Q.-H., and Wang, S. (2005). Final state of ecosystem containing grass, sheep and wolves with aging. *International Journal of Modern Physics C*, 16:177–190.

Hegselmann, R. and Flache, A. (1998). Understanding complex social dynamics: A plea for cellular automata based modelling. *Journal of Artificial Societies and Social Simulation*, 1(3).

Imai, K. and Morita, K. (1996). Firing squad synchronization problem in reversible cellular automata. *Theoretical Computer Science*, 165(2):475 – 482.

Jaffe, M. J., Leopold, A. C., and Staples, R. C. (2002). Thigmo responses in plants and fungi. *American Journal of Botany*, 89(3):375–382.

Kafeza, E. and Kafeza, I. (2009). Privacy issues in ami spaces. *Int. J. Netw. Virtual Organ.*, 6(6):634–650.

Kanada, Y. (1994). The effects of randomness in asynchronous 1d cellular automata (poster). *Artificial Life IV*.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467.

Kauffman, S. A. (1993). *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA.

Keenan, D. C. and O'Brien, M. (1993). Competition, collusion and chaos. *Journal of Economic Dynamics and Control*, 17(3):327–353.

Kirchkamp, O. (1995). Spatial evolution of automata in the prisoners' dilemma. Discussion Paper Serie B 330, University of Bonn, Germany.

Knospe, W., Santen, L., Schadschneider, A., and Schreckenberg, M. (2004). An empirical test for cellular automaton models of traffic flow. *Physical Review E*, 70(1).

Kobayashi, K. (1977). The firing squad synchronization problem for two-dimensional arrays. *Information and Control*, 34(3):177 – 197.

Komatsuzaki, T. and Iwata, Y. (2006). Modeling of sound absorption by porous materials using cellular automata. In Yacoubi, S. E., Chopard, B., and Bandini, S., editors, *ACRI*, volume 4173 of *Lecture Notes in Computer Science*, pages 357–366. Springer.

Komatsuzaki, T., Sato, H., Iwata, Y., and Morishita, S. (1999). Simulation of wave propagation using cellular automata. *Transactions of JSCES*, 1999:19990017.

Langton, C. G. (1990). Computation at the edge of chaos. *Physica D*, 42:12–37.

Lee, Y. C., Qian, S., Jones, R. D., Barnes, C. W., Flake, G. W., O'Rourke, M. K., Lee, K., Chen, H. H., Sun, G. Z., Zhang, Y. Q., Chen, D., and Giles, C. L. (1990). Adaptive stochastic cellular automata: theory. *Physica D*, 45(1-3):159–180.

Li, W. (1992). Phenomenology of nonlocal cellular automata. *Journal of Statistical Physics*, 68(5 / 6):829–882.

Li, W. and Packard, N. (1990). The structure of the elementary cellular automata rule space. *Complex Systems*, 4(3):281–297.

Li, W., Packard, N., and Langton, C. G. (1990a). Transition phenomena in CA rule space. *Physica D*, 45:77.

Li, W., Packard, N. H., and Langton, C. (1990b). Transition phenomena in cellular automata rule space. *Physica D*, 45:77–94.

Liebrand, W. B. and Messick, D. M. (1996). *Frontiers in Social Dilemmas Research*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Lumer, E. D. and Nicolis, G. (1994). Synchronous versus asynchronous dynamics in spatially distributed systems. *Phys. D*, 71(4):440–452.

Machin, C. H. C. (2002). Digital artworks: Bridging the technology gap. *Eurographics UK Conference, Annual*, 0:16.

Madore, B. F. and Freedman, W. L. (1983). Computer simulations of the belousov-zhabotinsky reaction. *Science*, 222:615–616.

Maerivoet, S. and De Moor, B. (2005). Cellular automata models of road traffic. *Physics Reports*, 419:1–64.

Maji, P., Ganguly, N., Saha, S., Roy, A. K., and Chaudhuri, P. P. (2002). Cellular automata machine for pattern recognition. In *ACRI '01: Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*, pages 270–281, London, UK. Springer-Verlag.

Marchese, F. T. (2006). The making of trigger and the agile engineering of artist-scientist collaboration. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, pages 839–844.

Margolus, N. (1984). Physics-like models of computation. *Physica D*, 10:81–95.

Mazoyer, J. (1987). A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183 – 238.

Mcculloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133.

Merrill, D., Kalanithi, J. J., and Maes, P. (2007). Siftables: towards sensor network user interfaces. In Ullmer and Schmidt (2007), pages 75–78.

Messick, D. and Liebrand, W. (1995). Individual heuristics and the dynamics of cooperation in large groups. *Psychological Review*, 102(1):131–145.

Min, H., Yi, S., Cho, Y., and Hong, J. (2007). An efficient dynamic memory allocator for sensor operating systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1159–1164, New York, NY, USA. ACM.

Mitchell, M., Hraber, P. T., and Crutchfield, J. P. (1993). Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130.

Moore, E. F., editor (1964). *Sequential Machines: Selected Papers*. Addison-Wesley Longman Ltd., Essex, UK, UK.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1(1):97–123.

Mynett, A. and Chen, I. (2004). Cellular automata in ecological and ecohydraulics modelling. In Sloot et al. (2004), pages 502–512.

Nagel, K. (1996). Particle hopping models and traffic flow theory. Working Papers 96-04-015, Santa Fe Institute.

Nagel, K. (2002). Cellular automata models for transportation applications. In *ACRI '01: Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*, pages 20–31, London, UK. Springer-Verlag.

Nagel, K. and Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *J. Phys. I France*, 2:2221–2229.

Negroponte, N. (1975). *Soft Architecture Machines*. MIT Press, Cambridge, MA.

Nehaniv, C. L. (2003). Evolution in asynchronous cellular automata. In *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, pages 65–73, Cambridge, MA, USA. MIT Press.

Nowak, M. A. and May, R. M. (1992). Evolutionary games and spatial chaos. *Nature (London)*, 359:826–829.

Nowak, M. A. and May, R. M. (1993). The spatial dilemmas of evolution. *International Journal of Bifurcation and Chaos*, 3:35–78.

Oliveira, G. M. B., de Oliveira, P. P. B., and Omar, N. (2001). Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. *Artif. Life*, 7(3):277–301.

Omohundro, S. (1984). Modeling cellular automata with partial differential equations. *Physica D*, 10:128–134.

Oono, Y. and Kohmoto, M. (1985). Discrete model of chemical turbulence. *Phys. Rev. Lett.*, 55(27):2927–2931.

Osterhout, J. K. (1994). *Tcl and the Tk toolkit*. Addison–Wesley, Reading, MA.

Paolo, E. A. D. (2000). Searching for rhythms in asynchronous random boolean networks. In Bedau, M., editor, *Alife VII: Proceedings of the Seventh International Conference*, pages 73–80. MIT Press.

Paolo, E. A. D. (2001). Rhythmic and non-rhythmic attractors in asynchronous random boolean networks. *Biosystems*, 59(3):185 – 195.

Popovici, A. and Popovici, D. (2002). Cellular automata in image processing. In *Proceedings of the 15th International Symposium on the Mathematical Theory of Networks and Systems*.

Preetham, A. J., Shirley, P., and Smits, B. (1999). A practical analytic model for daylight. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 91–100, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Puckette, M. S. (1996). Pure data: another integrated computer music environment. In *in Proceedings, International Computer Music Conference*, pages 37–41.

Puckette, M. S. (1997). Pure data. *Proceedings, International Computer Music Conference. San Francisco: International Computer Music Association*, pages 269–272.

Reas, C. and Fry, B. (2007). *Processing - A programming Handbook for Visual Designers and Artists*. The MIT Press. ISBN 0-262-18262-9.

Resnick, M. (1997). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds (Complex Adaptive Systems)*. The MIT Press.

Robert, F. (1986). Discrete iterations: A metric study. In *Series in Computational Mathematics*. Springer-Verlag.

Roli, A. and Zambonelli, F. (2002). Emergence of macro spatial structures in dissipative cellular automata. In *Proc. of ACRI2002: Fifth International Conference on Cellular Automata for Research and Industry*, volume 2493 of *Lecture Notes in Computer Science*, pages 144–155. Springer.

Ruffo, S. and Lio, P. (2001). *Dynamical Modelling in Biotechnologies*. World Scientific Publishing Company.

Sakoda, J. M. (1971). The checkerboard model of social interaction. *Journal of Mathematical Sociology*, 1:119–132.

Schadschneider, A. (2002). Cellular automaton approach to pedestrian dynamics - theory. *Pedestrian and Evacuation Dynamics*, pages 75–85.

Schelling, T. C. (1969). Models of segregation (in strategic theory and its applications). *American Economic Review*, 59(2):488–493.

Schelling, T. C. (1971). Dynamic models of segregation. *Journal of Mathematical Sociology*, 1(2):143–186.

Schreckenberg, M. and Sharma, S. D. (2002). *Pedestrian and Evacuation Dynamics*. Springer Verlag.

Settle, A. and Simon, J. (2002). Smaller solutions for the firing squad. *Theor. Comput. Sci.*, 276(1-2):83–109.

Shadbolt, N. (2003). Ambient Intelligence. *IEEE Intelligent Systems*, 18(4):2–3.

Shlesinger, M. F. (1992). New paths for random walkers. *Nature (London)*, 355:396 – 397.

Sloot, P. M. A., Chopard, B., and Hoekstra, A. G., editors (2004). *Cellular Automata, 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, Amsterdam, The Netherlands, October 25-28, 2004, Proceedings*, volume 3305 of *Lecture Notes in Computer Science*. Springer.

Sperber, M. (2001). Developing a stage lighting system from scratch. In *ICFP*, pages 122–133.

Stankovic, J. A., Lee, I., Mok, A., and Rajkumar, R. (2005). Opportunities and obligations for physical computing systems. *Computer*, 38(11):23–31.

Sutner, K. (1990). Classifying circular CA. *Physica D*, 45:386.

Taya, M. (2003). Bio-inspired design of intelligent materials. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5051 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 54–65.

Telewski, F. W. (2006). A unified hypothesis of mechanoperception in plants. *American Journal of Botany*, 93(10):1466–1476.

Thomas, R. and Organization., E. M. B. (1979). *Kinetic logic : a Boolean approach to the analysis of complex regulatory systems : proceedings of the EMBO course "Formal analysis of genetic regulation," held in Brussels, September 6-16, 1977 / edited by Rene Thomas*. Springer-Verlag, Berlin; New York.

Toffoli, T. (1984). Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D*, 10:117–127.

Toguchi, S., Akamine, Y., and Endo, S. (2008). Research into the generation of sound effects using a cellular automaton. In *ACRI '08: Proceedings of the 8th international conference on Cellular Automata for Reseach and Industry*, pages 323–328, Berlin, Heidelberg. Springer-Verlag.

Trifonova, A., Jaccheri, L., and Bergaust, K. (2008). Software engineering issues in interactive installation art. *International Journal of Arts and Technology*, 1:43–65(23).

Ulam, S. M. (1962). On some mathematical problems connected with patterns of growth of figures. In *Proceedings of the Symposia in Applied Mathematics*, pages 215–224.

Ullmer, B. and Schmidt, A., editors (2007). *Proceedings of the 1st International Conference on Tangible and Embedded Interaction 2007, Baton Rouge, Louisiana, USA, February 15-17, 2007*. ACM.

Umeo, H., Kamikawa, N., and Yunès, J.-B. (2009). A family of smallest symmetrical four-state firing squad synchronization protocols for ring arrays. *Parallel Processing Letters*, 19(2):299–313.

Umeo, H. and Yanagihara, T. (2009). A small five-state non-optimum-time solution to the firing squad synchronization problem - a geometrical approach. *Fundam. Inform.*, 91(1):161–178.

Vichniac, G. Y. (1990). Boolean derivatives on cellular automata. *Physica D*, 45(1–3):63–74.

von Neumann, J. (1966). *Theory of Self-Reproducting Automata*. University of Illinois Press, Urbana and London.

Waksman, A. (1966). An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66 – 78.

Weimar, J. R. (1997a). Cellular automata for reaction-diffusion systems. *Parallel Computing*, 23(11):1699–1715.

Weimar, J. R. (1997b). *Simulation with Cellular Automata*. Logos Verlag Berlin. ISBN 3-89722-026-1.

Wilson, P. R., Johnstone, M. S., Neely, M., and Boles, D. (1995). Dynamic storage allocation: A survey and critical review. pages 1–116. Springer-Verlag.

Winfree, A. T., Winfree, E. M., and Seifert, H. (1985). Organizing centers in a cellular excitable medium. *Physica D*, 17:109–115.

Wolfram, S. (1982). Cellular automata as simple self-organizing systems. *Caltech preprint CALT-68-938*.

Wolfram, S. (1983a). Cellular automata. *Los Alamos Science*, 9:2–21.

Wolfram, S. (1983b). Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644.

Wolfram, S. (1984a). Cellular automata as models of complexity. *Nature*, 311:419–424.

Wolfram, S. (1984b). Universality and complexity in cellular automata. *Physica D*, 10:1–35.

Wolfram, S. (1986a). Cellular automaton fluids: Basic theory. *Journal of Statistical Physics*, 45:471–526.

Wolfram, S. (1986b). *Theory and Applications of Cellular Automata*. World Press.

Wolfram, S. (1986c). *Theory and applications of cellular automata*. World Scientific, Singapore. ISBN 9971-50-124-4 pbk.

Wolfram, S. (1994). *Cellular Automata and Complexity: collected papers*. Addison-Wesley.

Wu, Q., Wang, Y., Cao, G., and Fei, Y. (2005). Locomotion control of distributed self-reconfigurable robot based on cellular automata. In Huang, D.-S., Zhang, X.-P., and Huang, G.-B., editors, *ICIC (2)*, volume 3645 of *Lecture Notes in Computer Science*, pages 179–188. Springer.

Wuensche, A. (1993). The ghost in the machine:basins of attraction of random boolean networks. *Cognitive Science Research Paper 281, University of Sussex, 1993.* to be published in Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity.

Wuensche, A. (1998). Discrete dynamical networks and their attractor basins. In *Complexity International*, pages 3–21.

Wuensche, A. (1999). Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter. *Complexity*, 4(3):47–66.

Xu, Y., Luo, F., and Wang, J. (2004). A new modeling method for elevator group control system with cellular automata. In *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*, volume 4, pages 3596–3599.

Yunes, J. B. (1994). Seven-state solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 127(2):313 – 332.

Zambonelli, F., Mamei, M., and Roli, A. (2002). What can cellular automata tell us about the behavior of large multi-agent systems? In Garcia, A. F., de Lucena, C. J. P., Zambonelli, F., Omicini, A., and Castro, J., editors, *SELMAS*, volume 2603 of *Lecture Notes in Computer Science*, pages 216–231. Springer.