





# Towards a Representation of Cellular Automaton Using Specification and Description Language

Pau Fonseca i Casas  

Universitat Politècnica de Catalunya, Barcelona 08034, CA, Spain  
pau@fib.upc.edu

**Abstract.** Environmental simulation is complex, not only due to the inherent complexity of the phenomenon that we are facing but also to the fact that the personnel involved in this kind of projects belongs to different areas and specialties. In this scenario, the use of a formal language is needed since it simplifies the interaction between the parts. A key element that must be represented in an environmental simulation model is a Geographical Information System (GIS) data. This representation often uses Cellular Automaton structures since it allows to represent, not only the data but also its behavior inside the simulation model. In this work, we explore the use of SDL, that among other benefits we can remark that it is an ITU-T standard language and allows a complete graphical description of the models and several tools allows a semi-automatic implementation of the models.

**Keywords:** SDL · Cellular automaton · Formal representation · Fibonacci function

## 1 Introduction

The data used on environmental simulation models often can be dynamically modified by the behavior of the model, and usually, the results of the simulation model are mainly this dynamic modification of the data. As an example, for a decision support system related to forest fires [1, 2], the data representing the temperature for a geographical area can be both an output from the model and an input to the model. Therefore, the data and its structure is a key element of the model definition. Focusing on the conceptualization of a simulation model, to be able to do a complete and non-ambiguous representation of the system is necessary to represent:

1. The structure: that allows depicting the hierarchical decomposition of the model and the relation between all the different subcomponents and sub-models.
2. Behavior: that details the model processes and activities.
3. Data: that detail, not only the data, but its relationship with the model, and how the nature of the data modifies the structure of the model itself. On the paper, data declarations are made using C notation, conform to Z.104 Annex C clause C.1 C language binding.

For environmental simulation, the problem with the data and its impact on the structure of the model is specifically how to represent a Cellular Automaton (CA), because CA are widely used to represent geographical and dynamical information in environmental simulation models [3–7].

Specification and Description Language (SDL) [8–10], is an ITU-T standard language, that allows a graphical, complete and unambiguous representation of a simulation model. The different concepts that the SDL covers are:

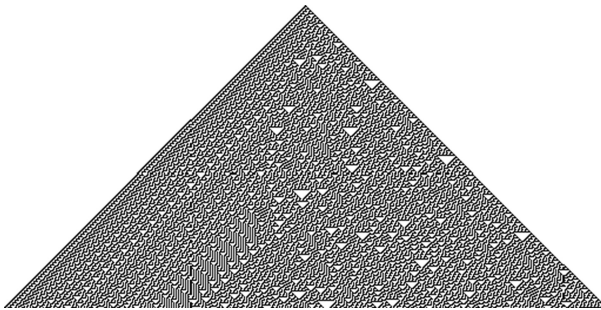
1. System structure: from the blocks to the processes and their related hierarchy.
2. Communication: signals, communication paths or channels, parameters that can be carried by the signals, etc.
3. Behavior: defined by different processes.
4. Data: based on Abstract Data Types (ADT).
5. Inheritance: useful to describe relations between objects and their properties.

In this paper we are focused on how SDL can be useful to describe the data related to an environmental model, using Inheritance, Data, and Communication in the diagrams.

### 1.1 Cellular Automaton

For the integration of a simulation model with Geographical Information System (GIS) data, that is often needed in an environmental model, it is useful to use a CA, due to its ability to effectively represent large-scale spatial dynamic phenomena [3, 4, 11].

CA is mainly a matrix and a set of rules that defines the matrix modifications over time. This behavior is completely specified in terms of a local relation. Table 1 shows one-dimensional CA after successive applications of the Wolfram’s 30<sup>th</sup> rule. Table 1 shows the 30<sup>th</sup> rule using Wolfram’s notation, see [12], where for each iteration we apply again the rule modifying the states of the cell.



**Fig. 1.** One-dimension CA following Wolfram’s 30th rule. Source Zhiming Wang [CC0]. Each row in the picture represents an evolution in the CA.

**Table 1.** Rule 30 CA.

Current pattern	111	110	101	100	011	010	001	000
New state for center cell	0	0	0	1	1	1	1	0

However, although this method of representing CA is simple, there are some clear limitations that we must consider:

1. Only one matrix (or vector in the rule 30 case) is considered, but in environmental simulation often there are several matrixes that are going to interact between them,
2. The evolution function that defines how the CA evolves is quite simple, the function can become more complex that cannot be represented with a simple pattern in a table. As an example of both limitations, in a simulation model representing a wildfire, it is needed to combine the data that represents the elevation, with the moisture, the wind direction, and other information, and calculate a complex function to be able to obtain a new state for the cell [2]. Also, on this function it is not considered how the time is going to be managed, there is no definition of the time needed to do the different operations (calculus).
3. The third constraint, that although is solved on our proposed CA is not detailed in this paper, is that we are restricted to a discrete state space (the matrix) while in environmental simulation, and specifically if we want to use continuous data (like the one represented on vector files i.e. representing rivers or territorial divisions), we must do always a rasterization of the data, losing some information in this process.

To solve these problems, we use a generalization of a cellular automaton that allows defining different layers on the same cellular automaton. We named this generalization  $m:n\text{-CA}^k$ , an initial proposal of this can be consulted on [13]. This generalization helps us to understand the complexity of the model we are going to face. Also, it simplifies the categorization of the different layers we are using, with a classification based on the existence or the absence of an intrinsic behavior on the layer. Main layers are those who have defined a specific behavior, this in our methodology is represented clearly because these layers have an SDL PROCESS representing this. Secondary layers are a simple matrix of data that are needed to perform a calculation, and because they do not own a specific behavior are fixed values.

## 1.2 Multi:N-Dimensional Cellular Automata ( $m:n\text{-CA}^k$ )

A multi:n-dimensional cellular automaton ( $m:n\text{-CA}^k$ ) is a generalization of a cellular automata composed by  $m$  layers with  $n$  dimensions each one, see (1).

$$m : n - CA^k \tag{1}$$

Where

- $m$ : is the automaton number of layers.
- $n$ : is the dimension of the different layers.
- $k$ : is the number of main layers (1 by default). If set to 0 we are using a matrix of cells, but no modification is applied to them.

A layer in an  $m:n\text{-CA}^k$  is a main layer if a transition function  $\Lambda$  is defined in order to modify its state. An  $m:n\text{-CA}^k$  automaton presents  $k$  main layers. Note that if  $k = 0$  then we have an  $m:n-1$ , that is just a matrix of data, if  $k = 1$  we have a usual CA (1:n-

CA is the same as an  $n$ -dimensional CA, a two-dimensional CA is represented as 1:2-CA). Some aspects to consider are:

1. Layers that modify their cell state are called **main layers**. The maximum number of main layers is  $m$ . The number of main layers is represented by  $k$  ( $m \geq k$ ). A given automaton may have more than one main layer. If  $k = 0$  we have just a matrix of data.
2. The **combination function**  $\Psi$  allows state calculation in a main layer, it depends on the state of all the other layers of the automaton.
3. It is not needed that the data follows the raster format because all layers share the **same reference system**. Thus, vector data may be used in  $m:n$ -CA <sup>$k$</sup> . The  $\Psi$  function determines the cell state independently of the structure of the layer data.

Vector data are quite usual in GIS, but in contrast to raster data (that is composed by a matrix containing the values), vector data presents a virtual continuous space that contains lines, polygons, etc. In this paper, we refer to vector data as an example of continuous information that can be used in a CA although in the examples we will be focused, for the sake of simplicity, on the raster case.

**Extension of the Definition of a Neighborhood and the Concepts of Vicinity and Nucleus.** In traditional cellular automata, the neighborhood function is defined to determine which cells are considered in the expression used to change the cell value, see Table 1. Because we accept vector data (continuous space) in our  $m:n$ -CA <sup>$k$</sup>  layer, the concept must be redefined without using cells and considering that all the layers share the same reference system, i.e. all the layers in a  $m:2$ -CA <sup>$k$</sup>  starts on the same physical position, as example (0,0). Therefore, the space that characterizes a neighborhood must be defined without cell dependency.

From a position  $x_1, \dots, x_n$ , the **vicinity** function defines the points to be considered in the evolution function in new layer-state calculation.

From a position  $x_1, \dots, x_n$ , the **nucleus** function defines the environment to be modified after the evolution function is calculated. The concept of neighborhood is related to the concept of topology and formalizes a colloquial concept.

In the mathematical definition, a topological space is a nonempty set  $X$  with a defined topology. It is represented as  $(X, T)$ . If  $(X, T)$  is a topological space and  $p$  is a point in  $X$ , a subset  $A$  of  $X$  is a neighborhood of  $p$  if an open  $U$  of the topology  $T$  exists such that  $p \in U \subset A$ .

The relationship between mathematical topology and the concepts of vicinity and nucleus allows us to formalize the ordination of points in layers on two levels. The first level represents the points considered in the calculation of a new state. The second level represents the points to be modified once the state changes.

The finest topology on  $X$  is the discrete topology, which implies the modification of points. The coarsest topology on  $X$  is the trivial topology, which consists of only two elements:  $T = \{\emptyset, X\}$ . In these two cases, the open sets that make up the space are defined by two topologies, nucleus and vicinity, which represent the points to be modified through a function  $\Lambda_k$ , named evolution function, that we will describe later, and the points to be considered in the calculation of a new state. Mathematical topology allows the explicit definition of neighborhoods for different points. Hence, in a raster

layer (discrete space), a neighborhood can be explicitly defined for each point. For  $m:n$ - $CA^k$  automata, these two topologies are defined as follows:

- **Vicinity topology** defines the set of points (neighborhood) of layer  $k$  to be considered in the calculation of  $\Lambda_k$ .
- **Nucleus topology** defines the set of points (neighborhood) of layer  $k$  to be modified by the calculation of  $\Lambda_k$ .

These two topologies define the neighborhood structures necessary for each point to establish the vicinity and the nucleus. However, not all neighborhoods can be used to represent the nucleus or the vicinity, and only one set can be used.

To define the set to be used for a point's neighborhood, a metric must usually be defined, based, for instance, on Euclidean distance (2).

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \quad (2)$$

Distance  $d(x, y)$  allows for the definition of neighborhood bases as follows (3)

$$B(x, r) = \{y \in \mathfrak{R}^m / d(x, y) < r\} \quad (3)$$

This is the usual topology for  $RxR$  [14] and will become one of the most common topologies for an  $m:2$ - $CA$  based on the  $RxR$  space defined by the usual distance, note that this is a continuous space. We can generally define a distance  $r$  from the point  $x$  by defining the **restrictions** of the selected neighborhood. A typical restriction rule is to calculate the minimum neighborhood that contains all points for which  $d(x, p) < r$ . For instance, in the usual topology presented in (3),  $B(x, r)$  is the minimum neighborhood that satisfies this restriction. In a more general topology, the restriction defines only one neighborhood for all the sets.

In an  $m:n$ - $CA^k$ , two restriction rules must be defined: one for the vicinity topology and one for the nucleus topology. These two restriction rules are used to construct the vicinity and nucleus functions. We can now define the vicinity and nucleus functions.

- **Vicinity function**  $vn_m(x_1, \dots, x_n)$  returns the minimum open set of the vicinity topology for the layer  $m$ , that contains point  $x_1, \dots, x_n$  and includes the maximum points that satisfy the restriction and the minimum points that do not satisfy the restriction. If the restriction is defined by the usual distance, it represents a neighborhood that contains the maximum points that satisfy  $d(t, p) < r$  and the minimum points that satisfy  $d(t, p) \geq r$ .
- **Nucleus function**  $nc_m(x_1, \dots, x_n)$  returns the minimum open set of the nucleus topology that contains point  $x_1, \dots, x_n$  and includes the maximum points that satisfy the restriction and the minimum points that do not satisfy the restriction. Depending on the type of data in the different layers, the topology nucleus and neighborhood are defined over  $N^n$  or  $Z^n$  (in the raster case) or  $R^n$  (in the vector case). We can consider however working in other systems like the Complex or the Octonions.

With this redefinition of the vicinity and nucleus central concept in CA, we can go further to understand how the CA is going to modify its values following a specified rule

**Combination Function ( $\Psi$ ).** Each cell of the matrix that defines an  $m:n\text{-CA}^k$  has a specific value. We define the number of possible states in the cell for the layer  $m$  with  $S_m$ , being a value that is not needed to be constrained in the body of the natural numbers, one can define  $S_m$  on the body of the Real or Octonions numbers (as an example) without any constraint. To combine the different  $S_m$  that belongs to the  $m:n\text{-CA}^k$  is needed to define a common reference, and a coordinate system, composed by  $n$  elements. With this, we can define the state of the cell in a  $m:n\text{-CA}^k$  as is presented on (4).

$$E_m(nc_m(x_1, \dots, x_n)) = S_i \tag{4}$$

The function  $E_m$  represents the state of each cell (nucleus) in the different layers of the automaton. Note that if one considers only the state for the main layers, we can note this with  $E_k$ . However, this state is not the global state of the automaton. For the coordinates  $x_1, \dots, x_n$  the function  $E_G$  returns the global state of the automaton. To be able to calculate this  $E_G$  is needed to define the Combination function ( $\Psi$ ), that returns the global state for a position using individual layer states, see (5).

$$\Psi\left(E_1(nc_1(x_1..x_n)),^{m-1}, E_m(nc_m(x_1..x_n))\right) = E_G(x_1..x_n) \tag{5}$$

The definition of the  $\Psi$  function depends on the structure of a given automaton. In a  $1:n\text{-CA}$ , this function is the identity function, returning as  $E_G$  the nucleus of the single layer that exists (in that case main layer). This can grow in complexity in other scenarios, see (6).

$$\Psi(E_1(nc_1(x_1..x_n))) = E_1(nc_1(x_1..x_n)) = E_G(x_1..x_n) \tag{6}$$

**Evolution function ( $\Lambda_k$ ) in an  $m:n\text{-CA}^k$**  In common cellular automata, the evolution function allows modify the  $E_m$  for each main layer, hence is focused on modify only the  $E_k$ , the main layers. Evolution function ( $\Lambda_k$ ) for a common cellular automata usually operate recalculating the  $E_k$  defining a  $\Delta t$  intervals, or, for a CA that does not define how to manage time, like the one presented on Fig. 1, calculating the new  $E_k$  from the previous  $E_k$  at a single step.

In  $m:n\text{-CA}^k$  automata, space can be represented as being continuous, but also time evolution can be considered as continuous, hence, the evolution function must also be (if needed) a continuous function. As we will see in our approach, the formal definition of the CA is based on SDL that will be agnostic on how the time is going to be updated, hence is possible to define an Activity Scanning [15] approach to model  $\Lambda_k$  achieving if needed a good approximation to a continuous time evolution. In the example proposed in this paper, is not needed to use continuous time, hence Event Scheduling usual approach will be enough.  $\Lambda_k$  is defined for main layer  $k$  to modify its state using the combination function  $\Psi$ .

The relationship between  $m:n$ -CA and common CA can now be established, being  $m:n$ -AC<sup>k</sup> a generalization of a common cellular automaton, since  $1:n$ -CA over  $Z_n$  defines a usual CA. We must establish a general method to define the CA structure and behavior ( $\Psi$  and  $\Lambda_k$  functions). To do so we explore the use of SDL in the next sections.

### 1.3 Specification and Description Language

Specification and Description Language (SDL) is a standard object-oriented formal and graphical language defined by the International Telecommunications Union–Telecommunications Standardization Sector (ITU–T) (the Comité Consultatif International Telegraphique et Telephonique [CCITT]) on the Z. 100 recommendation. On its origins, SDL was designed for the specification of event-oriented, real-time and interactive complex systems. These systems might involve different concurrent activities that use signals to perform communication. SDL is based on the definition of four levels to describe the structure and the behavior of the models: system, blocks, processes and procedures. In SDL *blocks* and *processes* are named *agents*. The outermost block, the *system* block, is an agent itself. Figure 2 shows these levels hierarchy.

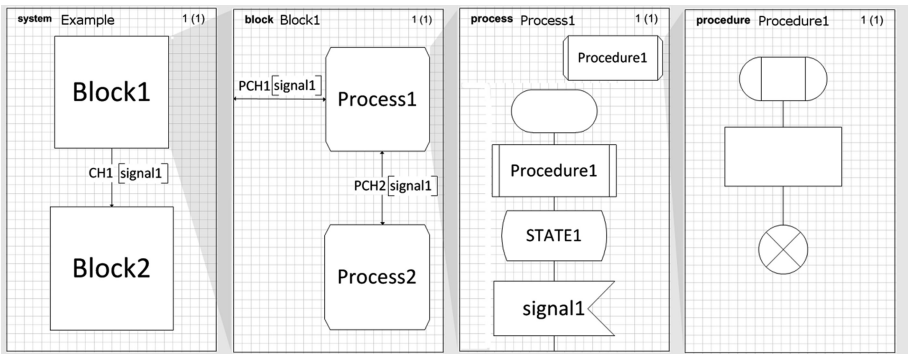
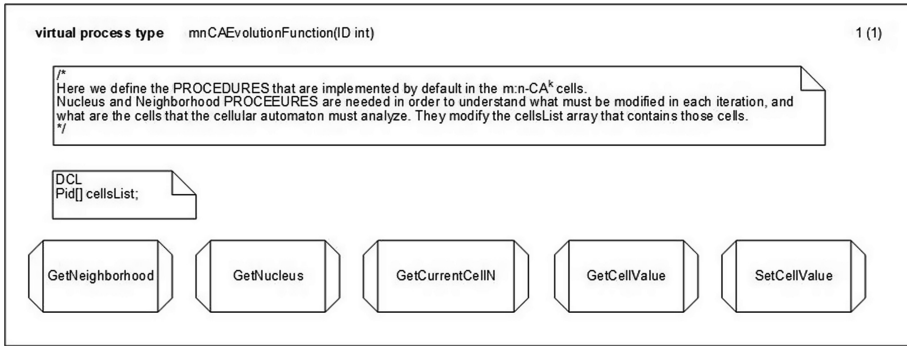


Fig. 2. A structural vision of an SDL model. 4 main different levels exist.

Although a textual SDL representation is possible (SDL/PR), this paper uses the graphical representation of the language (named SDL/GR). More details about the Specification and Description Language can be found in the recommendation Z.100 [16] or at the web site [17]. BLOCKS, PROCESS, and PROCEDURES define the basic structure and behavior of a simulation model, however, in order to represent environmental models, this is not enough. We need some structure in order to represent the data and its relationship with the simulation model, it is needed to detail, how the evolution of the simulation model modifies its surrounding data, and how this data influences on the model behavior. To do this we start with the definition of the cellular automaton following an  $m:n$ -CA<sup>k</sup> over  $N^k$  numbers.

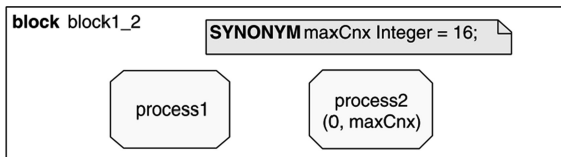
## 2 Representing m:n-CA<sup>k</sup> on SDL

In order to solve the representation of CA (and specifically m:n-CA<sup>k</sup>) with SDL we will define an AGENT TYPE to represent the layers of the CA, and a method to instantiate all the needed cells of the model, that will be represented also by a second AGENT TYPE. This allows representing graphically the interaction between the model and the data, and between all the layers that compose the CA. Starting with the needed information on each cell, we must define the neighborhood, the nucleus, the ID of the current cell, and define a method to modify the value of the matrix that contains the information of the CA, see Fig. 3. that represents the  $\Lambda_k$ . Note that the state of the CA cell, in the case of a main layer, is going to be modified due to the inherent behavior of the PROCESS, however, one must want to obtain the initial value from the matrix or to write this value to the matrix that represents this layer.



**Fig. 3.** Definition of the m:n-CA<sup>k</sup> on SDL. This PROCESS TYPE will define the nucleus, neighborhood and the needed PROCEDURES to work with a usual CA, as an example, by default a Moore neighborhood can be implemented. The specific behavior of the cell, the Evolution function, must be redefined for each specific case along with the PROCEDURES if needed.

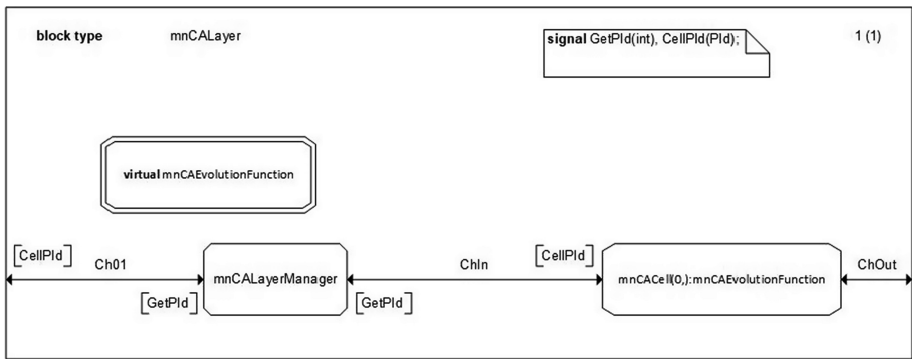
This process will be rewritten adding the behavior of the  $\Lambda_k$  that details how the cell behaves. Once the cell behavior is defined, the next step is to define its structure, mainly by several layers and cells. Again, this can be done using SDL agents. The layer is the element that (depending on the dimensions of the cellular automaton) creates all the needed cells. One can define this number of cells following a process like the one proposed in Fig. 4.



**Fig. 4.** A method to define the number of instances to be created, see [18]



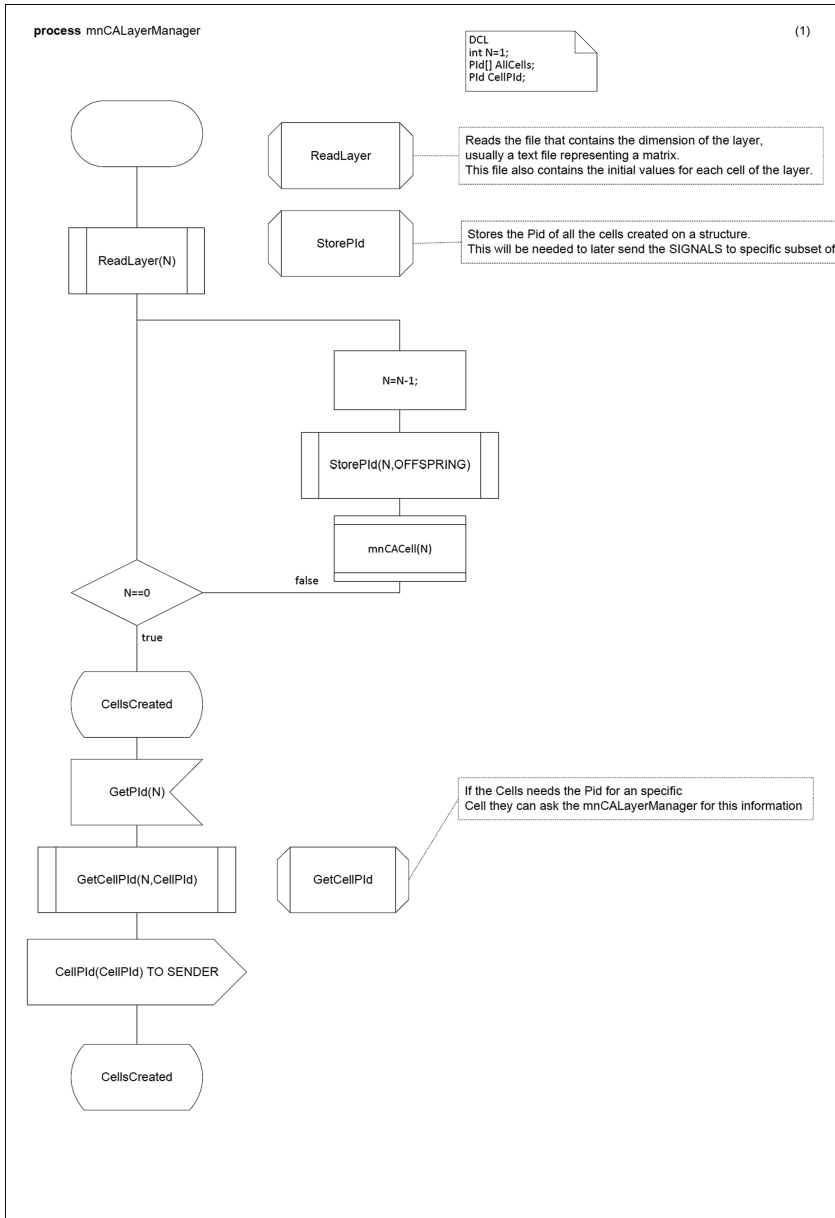
In that case, the modeler must define, for each layer the number of cells that compose this layer. We prefer to avoid using this approach since we are focused on CA modeling and each one of the different layers of our CA must be defined using a file that usually can contain the dataset to be used and modified during the execution of the model. We propose to use the approach shown in Fig. 5, where the dimensions of the layers (the number of cells) are obtained from the dataset (represented in the simple case in a text file), assuring a coherence between the dataset to be used in each layer in the CA and the conceptual model definition of the CA. Also, we can establish a relation between the cell number and the PId. With this information later we can simplify send SIGNALS to a subset of the cells that compose the layer.



**Fig. 5.** Definition of a m:n-CAk layer on SDL. In this BLOCK the *mnCALayerManager* creates all the instances of the cells; each instance receives its number (N) that identifies it on the matrix, obtained by reading the data file that contains the state for each cell at the initial state. We store for each N the PId in a table contained in the *mnCALayerManager*. ChIn will be the usual communication CHANNEL between the manager and the cells of the CA. ChOut will be used to send SIGNALS to the redefined evolution function, only if needed, due to the combination function represented on the layers diagram. Ch01 will be used in case the *mnCALayerManager* is asked by some AGENT to obtain the PId of a specific cell “N” of the layer (outside the layer).

We can add *mnCALayer* to a package named **mnCA** that will be used to simplify the definition of a CA in SDL. The cells do not have a specific behavior defined. The user must define the specific behavior for the cells as is represented in the example of this paper in Figs. 11 and 12. This is a key element since the modelers can focus on this diagram in order to understand cellular automaton behavior.

All these agents can be packaged and can be included in any project that needs to represent a cellular automaton using SDL. As an example, we present the well-known Game of Life using this, because of its simplicity. The main idea is that for the representation of a model that uses a cellular automaton it is only needed to write the behavior of the cells and the relation of the cells with other cells of other layers. All can be done graphically as we see next (Fig. 6).



**Fig. 6.** Definition of the mnCALayerManager. ReadLayer(N) PROCEDURE modifies N according to the number of existing layers on the CA.

### 2.1 Extending the SDL to Define a Cellular Automaton

When we try to define the behavior of the cellular automaton two main issues need to be solved, the time management and the multiple instances management.

Regarding the time management, we use the feature of SDL-2010 that allows defining time and priority in the SIGNAL. Every SIGNAL that is output has an optional parameter that defines the *time* needed to travel to its destination, and an optional parameter defining the signal *priority* with respect to other signal instances in the destination input queue scheduled, for the same time. From this time parameter and the value of **now** at the time the signal is output, an availability time is calculated. It is needed also to comment that the communication path may include delaying channels, so this delay must also be added to the calculus. If the availability time is greater than arrival time, the signal remains unavailable until the availability time is reached. SIGNAL instances in the input port are ordered by the time of arrival. If the time parameter is omitted, then the *delay* is zero; When a signal is *output* and no signal priority is specified, it is given the priority value 0. When there are several signals available with different signal priority values, the signal with the lowest priority value is selected. The signals in the input port are scanned in the following order to determine whether there is a signal that is enabled: first, by the order of the arrival time in the priority inputs, and then, by the order of the arrival time for other (non-priority) inputs. For those signals that have the same arrival time, the signal *priority* determines which signal is processed first. If two signals have the same signal *priority*, then the order is arbitrary. In Fig. 7 is represented a SIGNAL and how it will be used in the context of the paper.

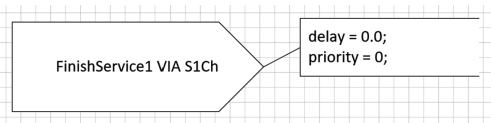


Fig. 7. Defining the delay, and the priority of the SIGNAL on SDL-2010.

The SIGNAL management in a CA differs from a usual process because all the cells of the nucleus will be updated by  $\Lambda_K$ , hence it will be usual to send SIGNALS to a set of cells (PROCESS). In order to simplify this, we propose to extend the semantics of SDL allowing to send a SIGNAL to a MNCA that will receive as a parameter a list of cell arguments. The cell list parameter can be also the keyword ALL, representing that all the cells of the layer will receive the SIGNAL. On Fig. 8 is shown an example where a SIGNAL is sent to the same AGENT (itself).

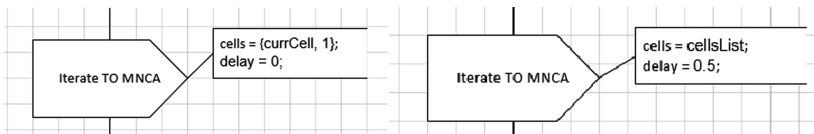


Fig. 8. Proposed extensions in order to define the cell that can receive a specific signal when the AGENT belongs to a cellular automaton. On the left side, we send the *Iterate* SIGNAL to two cells, one represented by the variable *currCell* and cell number 1. On the right side, we send the *Iterate* SIGNAL to all the cells on the cellist that *mnCAEvolutionFunction* owns by default.

On SDL one can send a SIGNAL to a list of destinations using “Iterate TO mnCACell[currCell] TO mnCACell [1]”, or defining a list of String <Pid> to be processed, but the proposed approach makes clear that the signal is sent to cells of the current CA (TO MNCA), simplifying its understanding. Also, notice that this extension can be implemented easily on SDL processing the signals to send one by one from the list but simplifies the lecture and the definition of the models; also, it allows to obtain improved codifications for this specific case.

## 2.2 The Game of Life

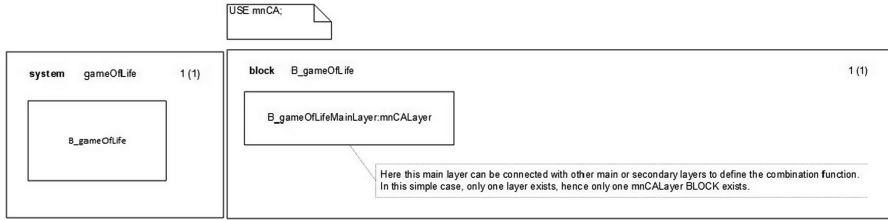
The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway in 1970 [19]. The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbors.

The rules that define the evolution of the automatons are:

- If the neighborhood of the cell contains less than two live cells, the cell dies.
- If the neighborhood of the cell contains more than three life cells die.
- If the neighborhood of the cell contains three life cells, the cell becomes a living cell.

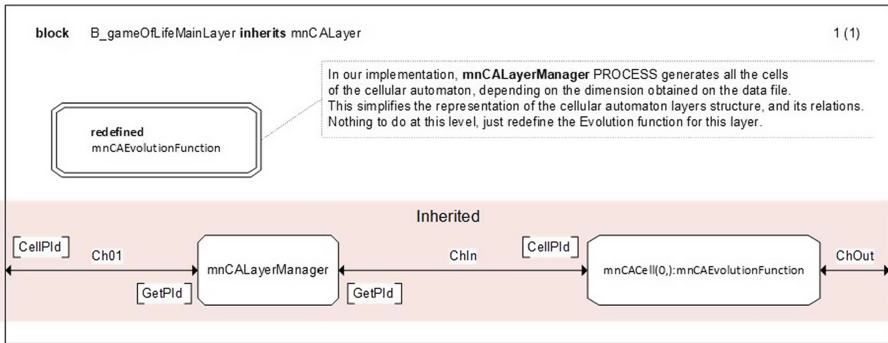
This description is not complete, since there is no description on the dimension of the CA, also, there is no description on the process followed by the CA that can cause different patterns to emerge. There is also no description on the time needed to do this modification, that can differ depending on the position. More interestingly, there is no method to connect with a specific dataset that contains an initial space of states of the CA or a method to connect with other models that will be using this CA. Although in this case (that is selected for the sake of simplicity, in order that the reader be used with the SDL representation), seems that the description of the CA following Wolfram’s of textual representation is simple, there are clear advantages on the formalization of the CA. Some examples where without a formalization one cannot represent the behavior of the CA can be reviewed here [5, 13, 20]. Also, notice that from this description a codification must be done, a process that can introduce some errors that must be verified.

The structure of SDL allows to detail if we want to execute each one of the different cells in different computers. This is a decision that can be represented graphically because in SDL all the elements contained inside a BLOCK agent can be executed in parallel. This is what happens in our case, since the element that details this is a BLOCK, see Fig. 9. Note that this diagram just represents the dimension and the structure of the CA.



**Fig. 9.** Definition of a cellular automaton layer on SDL that represents the Conway’s “Game of Life”. In this simple case, only one main layer exists, however in other cases several layers are going to interact, and at this level, the formal representation of this interaction will be needed. This interaction defines the combination function that represents how the information is going to flow from one layer to another. Here the SIGNALS that will transport the information between the layers will be defined.

Inside the BLOCK agent, we find (in the “Game of Life” case) a complete description of the behavior that rules the agent’s evolution. Since in a CA every cell behaves in the same manner, we only need to describe one single Process for each *Main Layer*. Thus, our *B\_gameOfLifeMainLayer* example Block will contain a single Process describing its nature, see Fig. 10. Notice that here one can represent the connection with other model elements that can interact with the CA during its execution.



**Fig. 10.** The definition of the cellular automaton layer. Here we find the main elements that compose the layer, the cells, and any other PROCESS that must be defined to represent the dynamic behavior of the cellular automaton. The inherited elements need not be redefined.

In Figs. 11 and 12 we can see a simple example of this representation. A cell has three possible states: *Loading*, *Death* and *Alive*. Note that the behavior of a cell is the definition of  $\Lambda_x$ , *Loading* state (Fig. 11) is used to initialize the CA, in the example, we assign the *Death* state by default with the exception of 5 concrete cells that will represent a Blister element. Once the model has been loaded, the PROCESS starts

iterating via the Iterate signal. Since an iteration represents a step in a CA, we easily can describe the evolution rules. In the example, we can see the evolution rules of Alive and Death state in Fig. 12.

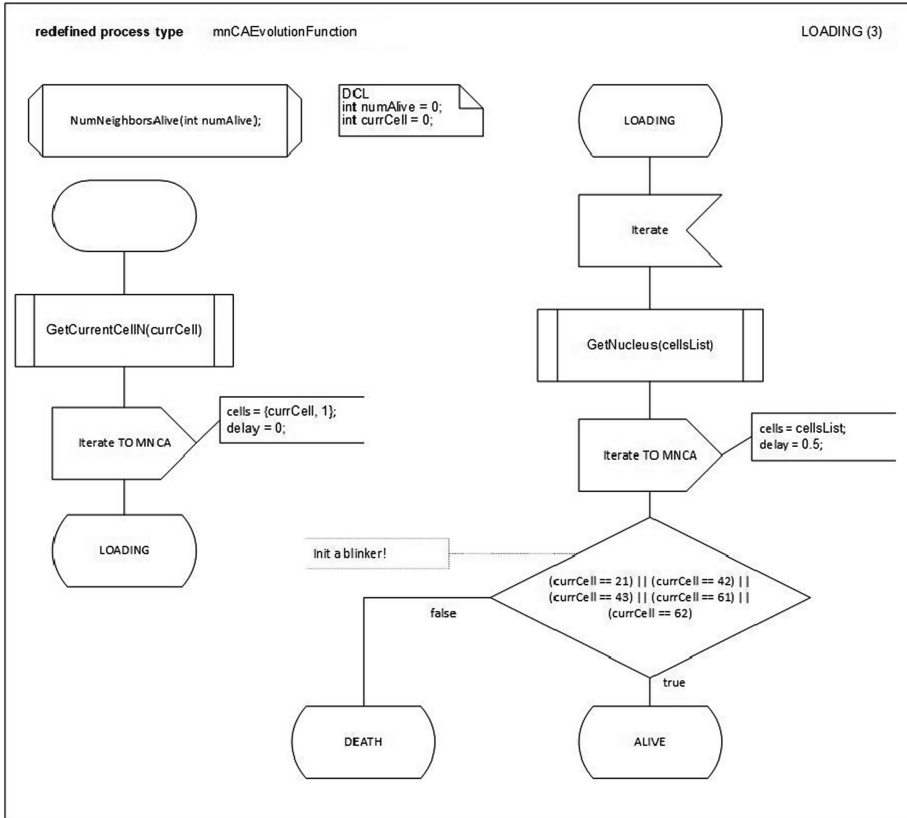
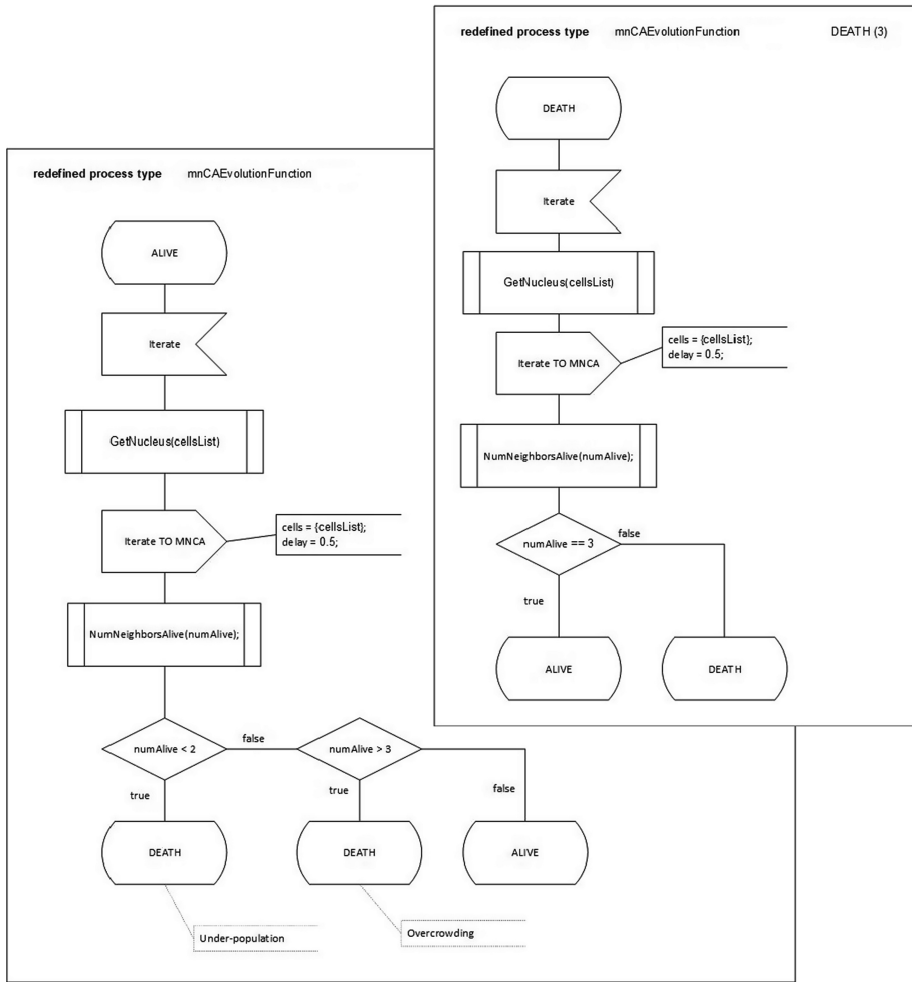


Fig. 11. Loading state for the *Game of Life* formalized using SDL.

To validate the model the experts can concentrate their efforts on the behavior described on the `mnCAEvolutionFunction`. Also, the implementation of the model can be based on the existing SDL tools. It is quite remarkable that the graphical definition we have of the model is complete and unambiguous. This model was successfully implemented on SDLPS [21, 22]. Some projects that implement this kind of solution are [1, 5, 13], where one can review that the definition encompasses not only the usual description of the rules of the CA, but also, the time needed to do the update on the cells, the relation of the data in each cell with other datasets, the relation of the CA with other models and the mechanism to combine the layers, among other elements that often are not represented in this kind of models.



**Fig. 12.** *Alive and Death states for the Game of Life formalized using SDL.*

### 3 Concluding Remarks

Environmental models need geographical information that often must be modified dynamically. CA are widely used to represent this information and connect it with a simulation model. Since the behavior of the cellular automaton is a key issue it is desirable that this behavior can be represented in an unambiguous and formal way. Some alternatives exist in order to formalize a CA, however, none of them is based on a complete, unambiguous, standard, graphical and formal language like SDL. This fact simplifies the verification process of a simulation model since the implementation can be done automatically by the tools that understand SDL.

In the paper we presented an extension of a CA that allows working with multiple raster and vector layers in a CA, extending the concept of nucleus and vicinity over a topological space. Based on this extension we define a new AGENT TYPE that allows representing CA structure. Also, it allows to automatically use the data sources automatically on the model defining a clean method to keep the dataset that represents each CA layer updated during the execution of the model.

The proposed extensions for SDL introduce the capability for SDL to become a language that can face problems related to the environment, where the representation of the landscape is a key aspect. This becomes more relevant in the frame of Industry 4.0, and considering that SDL can be a good candidate to become a key language in this area [23]. The approach simplifies the use of geographical data and CA models in a simulation model improving Validation and Verification processes. Modelers can see the different layers (sources of information) that compose the model in a graphical way. Also, in this graphical representation is represented the relations between all the model layers. In the *mnCALayer* this is clearly represented.

We showed a complete example, the Game of Life, to illustrate how SDL can represent Cellular Automata; however, this methodology can be used to represent real complex problems and take advantage of its graphical power, the unambiguity of the language, its completeness and the existing tools that allows an automatic validation, verification and generation of code.

A discussion arises regarding the computational complexity of a specific codification following this approach since each cell of the CA owns an AGENT. The codification that one can apply can simplify largely the computational resources needed and depends if finally, the platform will be a distributed or a sequential one, and in the case of a parallel architecture if it uses shared memory or not. This discussion, however, is a discussion that exist in the frame of CA codification (not for this approach), where the structure of the CA and the natural communication between all the different parts implies that often the codifications are resource-intensive, and can benefit from a clear and aseptic formal definition of the CA structure.

## References

1. Jové, J.F., Fonseca i Casas, P., Petit, A.G., Casanovas, J.: FireFight: a decision support system for forest fire containment (2014). [https://doi.org/10.1007/978-94-017-9136-6\\_19](https://doi.org/10.1007/978-94-017-9136-6_19)
2. Andrews, P.: BehavePlus fire modeling system: past, present, and future. In: Proceedings of 7th Symposium on Fire and Forest Meteorological Society (2007)
3. Benenson, I., Torrens, P.M.: Geosimulation. Wiley, Chichester (2004). <https://doi.org/10.1002/0470020997>
4. Andrews, G.: Cellular Automata and Applications, p. 29 (2008)
5. Fonseca, P., Colls, M., Casanovas, J.: A novel model to predict a slab avalanche configuration using m:n-CAk cellular automata. Comput. Environ. Urban Syst. **35**, 12–24 (2011). <https://doi.org/10.1016/j.compenvurbsys.2010.07.002>
6. Stephen, W.: Statistical-Mechanics-Cellular-Automata-Stephen-Wolfram-Article.pdf (1983). <https://doi.org/10.1103/RevModPhys.55.601>



7. Yue, H., Hao, H., Chen, X., Shao, C.: Simulation of pedestrian flow on square lattice based on cellular automata model. *Phys. A Stat. Mech. Appl.* **384**, 567–588 (2007). <https://doi.org/10.1016/j.physa.2007.05.070>
8. ITU-T: Specification and Description Language – Data and action language in SDL-2010 (2016)
9. ITU-T: Specification and Description Language – Overview of SDL-2010 (2011)
10. Doldi, L.: *SDL Illustrated - visually design executable models* (2001)
11. Wainer, G.A.: *Advanced Cell-DEVS modeling applications: a legacy of Norbert Giambiasi. Simulation* (2018). <https://doi.org/10.1177/0037549718761596>
12. Wolfram, S.: *A New Kind of Science* (2003)
13. Fonseca, P., Casanovas, J.: Simplifying GIS data use inside discrete event simulation model through M:N-AC cellular automaton. In: *International Mediterranean Modeling Multiconference, I3 M 2005, European Modeling Simulation Symposium EMSS 2005*, pp. 7–15 (2005)
14. Brendon, G.E.: *Topology and Geometry* (1993)
15. Law, A.M., Kelton, W.D.: *Simulation Modeling and Analysis* (1991)
16. ITU-T: Z.100. Specification and description language (SDL) (2016)
17. ITU-T: Specification and description language - overview of SDL-2010. <http://handle.itu.int/11.1002/1000/12846%0A>
18. Doldi, L.: *Validation of Communications Systems with SDL. The Art of SDL Simulation and Reachability Analysis* (2003)
19. Adamatzky, A., Durand-Lose, J.: *Collision-Based Computing*. Springer, London (2002). <https://doi.org/10.1007/978-1-4471-0129-1>
20. Fonseca i Casas, P., Colls, M., Casanovas, J.: Towards a representation of environmental models using specification and description language-from the fibonacci model to a wildfire model. In: *KEOD* (2010)
21. Fonseca i Casas, P.: Using specification and description language to define and implement discrete simulation models. In: *Summer Computer Simulation Conference, SCSC 2010 - Proceedings of the 2010 Summer Simulation Multiconference, SummerSim 2010*, pp. 419–426 (2010)
22. Fonseca i Casas, P.: *SDL distributed simulator*. In: *2008 Winter Simulation Conference* (2008). <https://doi.org/10.1109/WSC.2008.4736433>
23. Sherratt, E., Ober, I., Gaudin, E., Fonseca I Casas, P., Kristoffersen, F.: *SDL - the IoT language* (2015). [https://doi.org/10.1007/978-3-319-24912-4\\_3](https://doi.org/10.1007/978-3-319-24912-4_3)