Representing cellular automata using Specification and Description Language

Pau Fonseca i Casas
Universitat Politècnica de Catalunya
Jordi Girona 1-3
08034, Barcelona, Catalunya, SPAIN
(+34)934017732
pau@fib.upc.edu

Màxim Colls
Universitat Politècnica de Catalunya
Jordi Girona 1-3
08034, Barcelona, Catalunya, SPAIN
(+34)934017732
maximc@fib.upc.edu

Josep Casanovas
Universitat Politècnica de Catalunya
Jordi Girona 1-3
08034, Barcelona, Catalunya, SPAIN
(+34)934017732
josepk@fib.upc.edu

ABSTRACT

In this paper we show how to use Specification and Description Language (SDL) to represent cellular automata models. To achieve that we use a generalization of the common cellular automata, named m:n-CA^k, allowing the definition of multiple layers in a single cellular automata. Also we add some extension to SDL language to simplify the representation of these automata. Thanks SDL and m:n-CA^k the behavior of the cellular automata model can be defined in a graphical way allowing the complete and unambiguous description of the simulation model that uses it.

SDL is a modern object oriented formalism that allows the definition of distributed systems. It has focused on the modeling of reactive, state/event driven systems, and has been standardized by the International Telecommunications Union (ITU) in the Z.100.

In our approximation the SDL representation of the model (a Microsoft Visio® diagram) can be executed. This implies that since is not needed to perform any implementation, the verification process is simplified.

The proposed methodology (and infrastructure) allows the definition (and implementation) of a simulation model that uses cellular automata following the Specification and Description Language.

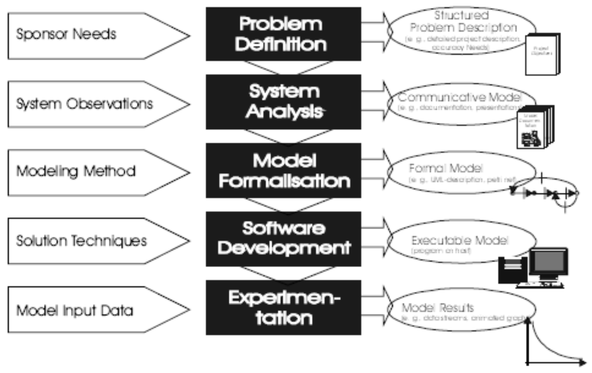


Figure 1: The formalization of the model can be considered as a product itself [1].

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Physical sciences and engineering – *Mathematics and statistics*

General Terms

Documentation, Design, Human Factors, Standardization, Languages, Theory.

Keywords

SDL, Simulation, Formalisms, cellular automata.

1. INTRODUCTION

The construction of a simulation model sometimes lacks in the formalization process needed to understand the model before any implementation. This model behavior understanding helps in the implementation process and in the communication between the different personnel involved in the model construction. Also can be considered a product itself [1].

Different formalisms exist in order to represent a simulation model, like Petri Nets or DEVS among others.

Some tools have been build in order to allows help the model implementation from the specification [2], [3] and some allows the distribute execution of the models like CD++[4].

2. SDL LANGUAJE

SDL is the acronym of Specification and Description Language. SDL is an object-oriented, formal language defined by the International Telecommunication Union – Telecommunication Standardization Sector (ITU–T) (formerly *Comité Consultatif International Télégraphique et Téléphonique* [CCITT]) as Recommendation Z.100 [5]. The language is designed to specify complex, event-driven, real-time, interactive applications involving many concurrent activities using discrete signals to enable communication.

SDL is a powerful and modern language widely used in different areas, not only in simulation area. It has been standardized by the International Telecommunications Union (ITU) in the Z.100, and can be used easily in combination with UML.

The definition of the model is based on different components:

- **Structure**: system, blocks, processes and processes hierarchy.
- **Behavior**: defined through the different processes.
- **Data**: based on Abstract Data Types (ADT).
- **Communication**: signals, with the parameters and channels that the signals use to travel.

- **Inheritances**: to describe the relationships between, and specialization of, the model elements.

The language has 4 levels (i) System, (ii) Blocks, (iii) Processes and (iv) Procedures, as we can see in the next figure.

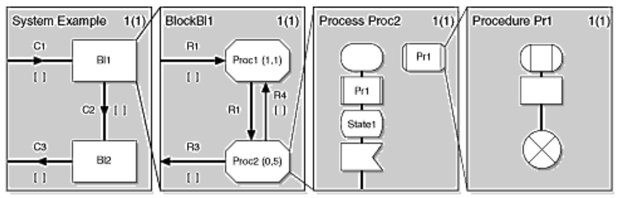


Figure 2: SDL levels, (http://www.iec.org/online/tutorials/sdl/topic04.html)

To know more about SDL the recommendation Z.100 [5] can be consulted, also a lot of information can be reviewed in the www.sdl-forum.org website.

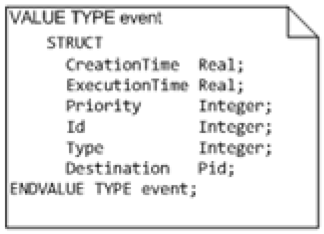
In our domain the signals are equivalent to the events that rule the modification of the states of the different elements, for that in this paper the use of event or signal is equivalent (see the section **Error! Reference source not found.** to review a discussion of this).

3. SDLPS OUR SDL IMPLEMENTATION

To implement our models we can use different existing tools that understand SDL language, like Cinderella [6] or Telelogic [7] of IBM. We develop our tool in order to improve the existing solutions adding some new capabilities:

- Allow to work with the delaying signals (as we can see below).
- Allow to work with cellular automata.
- Allow to work with intelligent agents.
- Allow a distributed simulation of the models.
- For these reasons we decide to implement our tool named SDLPS [8].

In SDLPS all the signals can carry the parameter defined in the structure represented in the Figure 3.



```
VALUE TYPE event
    STRUCT
      CreationTime   Real;
      ExecutionTime  Real;
      Priority       Integer;
      Id             Integer;
      Type           Integer;
      Destination    Pid;
ENDVALUE TYPE event;
```

Figure 3: Structure related to the SDLPS signals.

ExecutionTime, of **event** structure, allows defining the time when the signal (carrying the event) must reach its destination. Other elements are: (ii) *Priority*, the priority of the event, used to break a possible simultaneity of events. (iii) *CreationTime*, representing the time when the event is created. (iv) *Id*, an identifier of the event. (v) *Time*, the clock of the process. (vi) *Destination*, the final destination (process PId) of the signal. Not all the parameters of **event** structure must be defined, only those needed to fully define the behavior of the model.

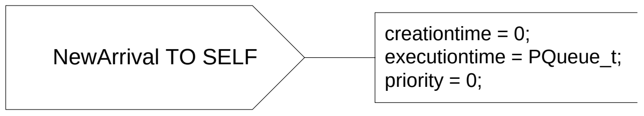


Figure 4: Defining the executionTime, and other parameters, of the signal using the SDL time extensions.

These extensions are now under discussion on the ITU-T Study Group 17 (http://www.itu.int/ITU-T/studygroups/com17/index.asp) to be included in the next release of the standard (SDL-2010).

In our example, since the Fibonacci model does not depend on time this is not used.

4. CELLULAR AUTOMATA

Cellular automata are discrete dynamical systems whose behavior is completely specified in terms of a local relation [9].

One-dimensional cellular automata are based in a row of "cells" and a set of "rules". A two-dimensional cellular automata uses rectangular grids of cells.

Each one of the different cells can be in one of different "states" (the number of possible states depends on the automata). Thinking states as numbers, in a two-state automata, each cell can be only in 1 or 2 state.

Cells represent automata space; time advances in discrete steps following "the rules", the laws of "automata universe", usually expressed in a small look-up table. At each step every cell computes its new state in function of its closer neighbors. Thus, system's laws are local and uniform.

Next figure shows one-dimensional cellular automata initial state and successive two states after rules application.

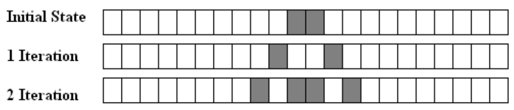


Figure 5:One-dimensional cellular automata

Since we are working with an extension of the common cellular automata, first we must define this extension.

5. M:N-CAK CELLULAR AUTOMATA

The main objective of m:n-CAk is to simplify the use of geographical and environmental information in a simulation model. As a result, m:n-CAk cellular automata is defined. Mainly is a generalization that simplifies the use of raster and vectorial data in a simulation model, and also extends the definition of common cellular automaton over the Topology mathematical concept.

m:n-CA^k was first defined on [10], in the next lines we describe the more important aspects of this cellular automaton generalization.

First said that m:n-CA^k means multi n-dimensional cellular automaton. Mainly is a generalization of cellular automata defined as follows:

Definition 1: m:n-AC^k

A multi n dimensional cellular automaton is a cellular automaton generalization composed by m layers with n dimensions each one.

The representation is:

$$m.:n-AC^k$$

Where

m: is the automaton number of layers.

n: is the different layers dimension.

k: is the number of main layers (1 by default).

With this notation, a usual two dimensional cellular automaton is represented by a 1:2-AC. A transition in a m:2-AC cellular automata is defined as in a 2-dimensional cellular automata, but main layer cell state is a combination of data contained in the m-1 secondary layers at the same position.

A m:n-AC implements a set of rules determining its transitions, but includes some new features:

- Layers modifying its cells values (states) are named main layers. Main layers maximum number is m. Number of main layers are represented by k ($m \geq k$). More than one main layer can exist in the same automata.
- Combination function Ψ allows state calculation in a main layer depending on state in other layers of automata.
- It is not necessary, that contained data follows raster structure since all layers are referenced in a coordinate system (usually georeferenced). This allows the use of vectorial data in cellular automata. Ψ function determines cell state independently of layers data structure.

Definition 2: $E_m[x_1,..,x_n]$, layer m state in $x_1,..,x_n$ position

E_m is a function describing cell state in position $x_1,..,x_n$ of layer m.

E_m function allows state representation for each cell in the different layers of the automata, but this is not the global state of the automata. This state is represented by the *EG* function.

Definition 3: $EG[x_1,..,x_n]$, automata status in $x_1,..,x_n$ position.

EG returns automata global state in position referenced by coordinates $x_1,..,x_n$.

The global state of cellular automata depends on EG function in all automata positions.

Combination functions Ψ is represented by equation:

$$\Psi(E_1[x_1..x_n], \overset{m-2)}{\cdot\cdot\cdot}, E_m[x_1..x_n]) = EG[x_1..x_n] \quad (1)$$

Definition purpose for Ψ function depends on automata structure, but in a 1:n-AC this function is the identity function.

$$\Psi(E^1[x_1..x_n]) = EG[x_1..x_n] \text{ ssi k=1} \qquad (2)$$

Global state of the cellular automata only depends on the state of main layer. This is the usual case in the common cellular automata with only one layer.

5.1 EVOLUTION FUNCTION (Λ) IN A M:N-AC

In a common cellular automaton, evolution function allows global automata state change through cells value modification.

In a m:n-AC^k vectorial layers use makes necessary to generalize the neighborhood and later define a new function that determines something similar to cell size (nucleus function). These two functions are defined in detail in next subsection.

m:n-AC^k have the capability to represent continuous space. In order to model continuous time, evolution function must be a continuous function, too.

Evolution function it's only defined in layers that modify its state (as we see previously named main layers).

Definition 4: Main layer

A layer in a m:n-AC is a main layer if a transition function Λ is defined in order to modify its state.

Now we can define a m:n-AC^k depending on the number of included main layers.

Definition 5: m:n-AC^k

A m:n-AC automaton only presents one main layer, an m:n-AC^k automaton presents k main layers.

Common case is a m:n-AC^1, an automata with only one main layer, therefore a m:n-AC automata.

Definition of the evolution function depends on the layer that modifies.

Definition 6: Evolution Function Λ_m

Function defined for the layer m to modify its state through the state of others layers using combination function Ψ, and vicinity and nucleus functions.

Intuitively evolution function allows the representation of the modifications in this layer (modifications in nucleus area of a point $x_1.x_n$), using the state of other layers with combination function Ψ, and the vicinity area.

5.2 NEIGHBORHOOD DEFINITION EXTENSION, VICINITY AND NUCLEUS CONCEPT.

In a traditional cellular automaton neighborhood function must be defined in order to determine cells to be considered in the evolution function.

In a m:n-AC^k, due to the vectorial representation capability is necessary to redefine the concept without using cells with the help of some kind of reference system.

Therefore is necessary to define the space area characterizing neighborhood without cell dependency. **Vicinity** function defines, from a position $x_1,..,x_n$, the points to be considered inside evolution function (to calculate the new state of a cell). **Nucleus** function allows to define, from a position $x_1,..,x_n$, the environment to be modified after evolution function calculation (the cells to be modified after the calculus, or the space in the continuous case). Neighborhood concept is related to topological concept formalizing a colloquial concept. Remembering topology mathematical definition, a topological space is a non empty set X with a defined topology. Representation is (X, T). If (X, T) is a topological space and p a point of X, a subset "A" of X is a neighborhood of p if and open U of the topology T exist as $p \in U \subset A$.

Existing relation between mathematical topology and vicinity and nucleus concepts allows formalizing point's ordination in layers in two levels:

1. First level represents points to be considered in order to calculate new state.
2. Second level represents points to be modified once state changes.

Finest topology on *X* is discrete topology imply punctual modification. The coarsest topology on *X* is a trivial topology, composed by only two elements: T= {Ø, X}.

In these two cases open sets configuring space are defined by two topologies: nucleus and vicinity, representing points to be modified through Λ_m function and points to be considered in new state calculus.

Mathematical topological concept permits explicit definition of neighborhoods for different points. Hence for a raster layer (discrete space) the neighborhood for each point can be explicitly defined.

Definitions of these two topologies for m:n-AC automatons are:

Definition 7: vicinity topology

Topology defining the set of points (neighborhood) for layer m, to be considered for Λ_m calculus.

In a similar way for nucleus topology:

Definition 8: nucleus topology

Topology defining the set of points (neighborhood) for layer m, to be modified by Λ_m calculus.

These two topologies define neighborhood structures necessary for each point in order to establish vicinity and nucleus. In despite of that, not all neighborhoods can be used to represent nucleus or vicinity, and only one set can be used.

To define the set to be used from a point neighborhood, usually can be necessary to define a metric, based for instance in Euclidian distance:

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \qquad (3)$$

Distance d(x,y) allows the definition of neighborhood bases as:

$$B(x,r) = \{ y \in \Re^m / d(x,y) < r \} \qquad (4)$$

This is the usual topology on RxR [11], can be one of the more indicated topologies for a m:2-AC based in the RxR space defined the usual distance.

In a general way we can define a distance r from the point x defining **restrictions** of the selected neighborhood.

A typical restriction rule can be to calculate minimum neighborhood containing all the points accomplish $d(x,p)<r$. For instance, in the usual topology presented in (4), B(x,r) are the minimum neighborhood to accomplishes this restriction. In a more general topology, like the one presented in next figure, restriction defines only one neighborhood from all the sets. In an m:n-AC^k two restrictions rules have to be defined, one for the vicinity topology, and other for nucleus topology. These two restriction rules are used to construct vicinity and nucleus functions. Now we can define vicinity and nucleus functions.

Definition 9: Vicinity function $vn(x_1,..,x_n)$

Function returning minimum open set of vicinity topology containing point $x_1,..,x_n$, and including maximum points that accomplishes the restriction and minimum points not accomplishing the restriction.

If the restriction is defined through usual distance its represents neighborhood containing maximum points that accomplish $d(t,p)<r$ and minimum points that accomplish $d(t,p) \geq r$

In the same way we define the nucleus function.

Definition 10: Nucleus function $nc(x_1,..,x_n)$

Function returning minimum open set of nucleus topology containing point $x_1,..,x_n$, and including maximum points that accomplishes the restriction and minimum points not accomplishing the restriction.

Our main purpose is to represent graphically and as simple as we can these definitions, allowing an automatic simulation of the represented model through SDLPS. Also, thanks the use of m:n-CA^k GIS data can be represented easily in different layers of the m:n-AC^k. Suitable data are vectorial data (2DLayers) or raster data (3DLayers). All layers must be referenced in a coordinate system (usually georeferenced) [12]. An example of use of m:n-CAk over continuous space can be reviewed in [10].

6. SDL EXTENSIONS

In order to simplify the representation of the m_n-CA^k cellular automata we add some extensions to SDL language to improve the readability of the model.

First of all, note that in cellular automata, all the cells have the same behavior. This implies that is not needed to represent all the cells, but only one cell. Also is needed to represent the relation that have with the neighborhood (that of course is specific of the cellular automata), and the relation with the other layers. That means define the **combination**, **vicinity** and **nucleus** functions.

In SDL language we can use **types** to define blocs that have the same behavior (as is usual in any OO language).In order to represent the communication between the different cells, are needed to represent all the communication channels between the cells. This implies the need of represent at least a set of cells that belongs to the vicinity (and the nucleus) of a cell. Also if we want a complete description of the cellular automaton (if we want an automatic generation of code), we need to represent all the cells of the cellular automaton.

In order to simplify the representation (avoiding the representation of all the cellular automaton cells) we decide to add a new kind of agent to SDL language, the **mnca**. This agent have the same behavior as the agent block, with a particularity, **mnca** agent is defining the entire cellular automaton, but only is needed to represent one cell. This implies that is needed to use a declarations section that defines the structure of the cellular automata (mainly the dimension of the cellular automata, and the size of each one of these dimensions). You can see a declaration of the cellular automaton in the **Figure 8**.

Also, since we want to avoid the representation of all the channels needed to send and receive the information related to consult the values in other cells (and since now the **mnca** agent represents in fact the whole cellular automaton although we only details the behavior of one cell), is needed to implement the next five methods.

- **mncaGetCurrentCell**(int MNCA_CURR_CELL): helps to obtain the index of the current cell.
- **mncaGetCellValue**(int MNCA_GET_CELL, int MNCA_GET_VALUE): helps to get the value of an specific cell, indexed by MNCA_GET_CELL.
- **mncaSetCellValue**(int MNCA_SET_VALUE): to set the value in a specific cell.
- **mncaGetLayerValue**(string MNCA_LAY, int MNCA_LAY_CELL, int MNCA_LAY_VALUE): to get the value of a cell of other layer.
- **mncaSetLayerValue**(string MNCA_LAY, int MNCA_LAY_CELL, int MNCA_LAY_VALUE): to set the value of a cell of other layer.

Since **mnca** block represents all the cells of the cellular automaton, is needed to define how send a signal from one cell to other cell of the same **mnca**.

To do this we are using an extension of the language that allows completing the definition of a signal (Figure 6).

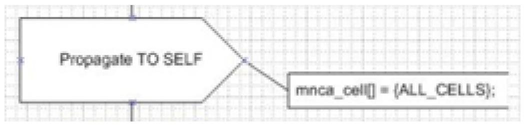


Figure 6: Sending a signal from on cell to other in the same mnca block.

First, note that the signal is send to the same element (TO SELF), as is defined in the standard. In order to distinguish between the different cells represented in the **mnca** SDL agent we are using the extension mnca_cell[]={cells} that defines the cells of the **mnca** block that receives the signal.

We can define an array of cells (the cells that can receive the signal). Also we can use ALL_CELLS, to send the signal to all the cells of the **mnca** agent.

6.1 GET PARALLEL

At this point is needed to mention that the current extension of the Specification and Description Language do not add more expression power to the language, but allows a clear representation of the diagrams. The implementation of these methods in SDLPS simplifies the definition of the model and improves its readability. Also remark that since in SDL language all the blocks (that means in our case cells and layers of a cell) can be executed in parallel, using our extension this can be achieved too. This implies that each one of the different cells and each one of the different layers belonging to the **mnca** agent can be executed in different machines.

For extension reason is not possible to explain in detail here the implemented solutions done. Only remark that at this point of the development the system is working with a conservative algorithm [13]. The system, named SDLPS [8], uses a manager to assign the resources (IP's of the machines that can be used for the simulation) at the beginning of the execution. At this point this assignment of resources is static.

7. CELLULAR AUTOMATA SDL REPRESENTATION

As an example we represent cellular automata that calculate the well known Fibonacci function. First is needed to define the different elements that compose our simulation model.

This first level of the SDL diagrams, in this case, only contains a single block, representing the cellular automata that implements the Fibonacci function.

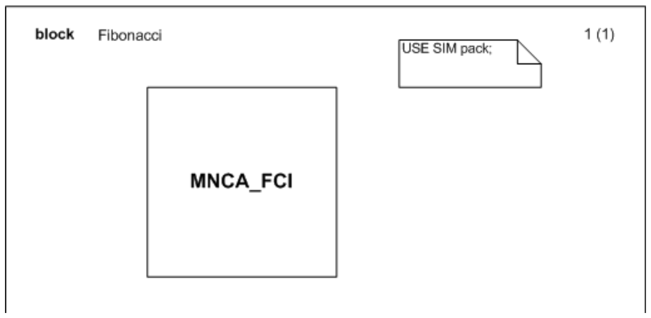
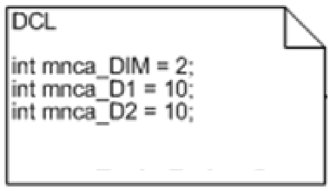


Figure 7: Fibonacci cellular automata.

This first level of the model helps to understand the different elements that must be combined inside the model.

Next, we must define the structure of this m:n-CAk cellular automata. First is needed to define the number of cells (the dimensions) that the cellular automata have. This is done with a DCL (declaration block):



```
int mnca_DIM = 2;
int mnca_D1 = 10;
int mnca_D2 = 10;
```

Figure 8: DCL block with the definition of the structure of the cellular automata.

On this block, the mnca_DIM variable defines the number of dimensions of the cellular automata, and mnca_D1 to mnca_Dn defines the size of each one of these dimensions. In that case we have a matrix (10x10).

Also we need to declare the different functions that rules the behavior of the cellular automata, these functions are:

- mncaVicFunc: defines the vicinity functions, as we can see next (**Figure 12**).

- mncaNucFunc: defines the nucleolus function, as we can see next (**Figure 13**).

In the next figure we can see the SDL representation of the cells of our m:n-CA^k cellular automata. At this level we can see the signals (carrying the simulation events) that are sending from each one of the different layers to other layers in order to establish the communication. This is not the case in this example since we have only one layer. Looking this structure the m:n-CA^k cellular automata used here is 1:2-CA over N.

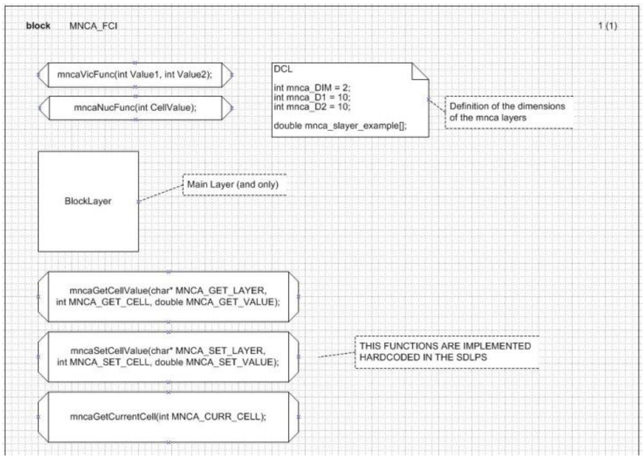

block MNCA_FCI
1 (1)
mncaVicFunc(int Value1, int Value2);
mncaNucFunc(int CellValue);
DCL
int mnca_DIM = 2;
int mnca_D1 = 10;
int mnca_D2 = 10;
double mnca_slayer_example[];
Definition of the dimensions of the mnca layers
BlockLayer
Main Layer (and only)
mncaGetCellValue(char* MNCA_GET_LAYER, int MNCA_GET_CELL, double MNCA_GET_VALUE);
mncaSetCellValue(char* MNCA_SET_LAYER, int MNCA_SET_CELL, double MNCA_SET_VALUE);
THIS FUNCTIONS ARE IMPLEMENTED HARDCODED IN THE SDLPS
mncaGetCurrentCell(int MNCA_CURR_CELL);

Figure 9: m:n-CA^k cell representation. Here we can see the relation between the different layers.

In the *BlockLayer* we can define different processes that can be executed sequentially. In our case only one process is defined as we can see in the next diagram.

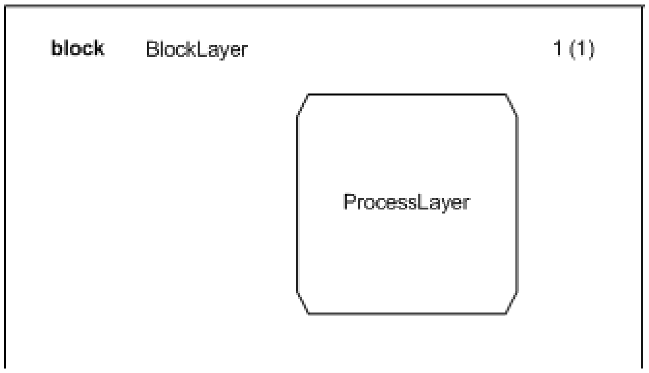


Figure 10: Representation of the layer.

From this complete representation of the cellular automata structure we can go further to define the behavior. In the next diagrams we show the processes and the procedures that defines the *Evolution* function, the *Vicinity* function and the *Nucleus* function of the cellular automata.

Evolution function is defined in the *ProcessLayer*, in **Figure 11**.

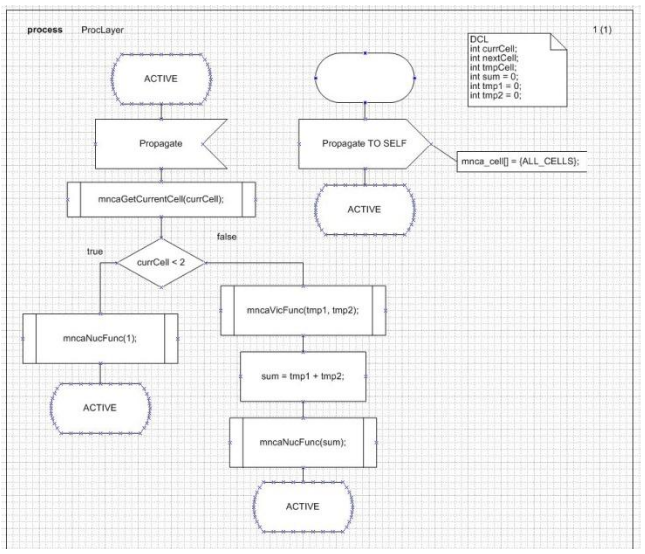


Figure 11: representation of the behavior of the cell.

Once the behavior of the evolution function of the cellular automata is defined, we need to explain how the *Vicinity* function and a *Nucleus* function behaves. These two functions are used in the definition of the behavior of our cellular automaton (*ProcessLayer* process).

Vicinity function defines a neighborhood with the visibility of the cellular automata. *Nucleus* function defines what is the space (cells in the discrete case) that must be modified once the *evolution* function is executed.

We characterize the representation of the *Vicinity* and *Nucleus* function for our Fibonacci model in **Figure 12** and **Figure 13**.

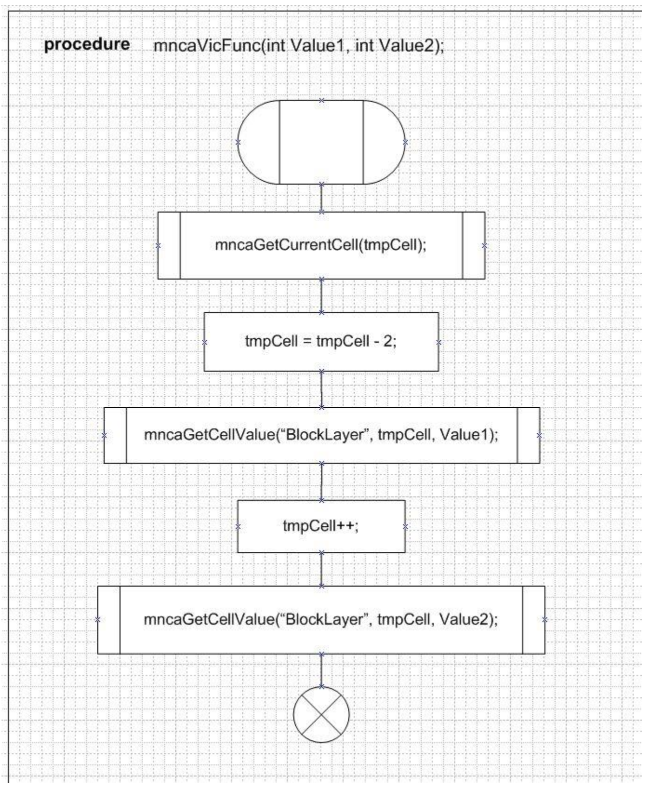

procedure mncaVicFunc(int Value1, int Value2);
mncaGetCurrentCell(tmpCell);
tmpCell = tmpCell - 2;
mncaGetCellValue("BlockLayer", tmpCell, Value1);
tmpCell++;
mncaGetCellValue("BlockLayer", tmpCell, Value2);

Figure 12: Vicinity function, representing the cells that must be consulted to modify the value of the cellular automata nucleus.

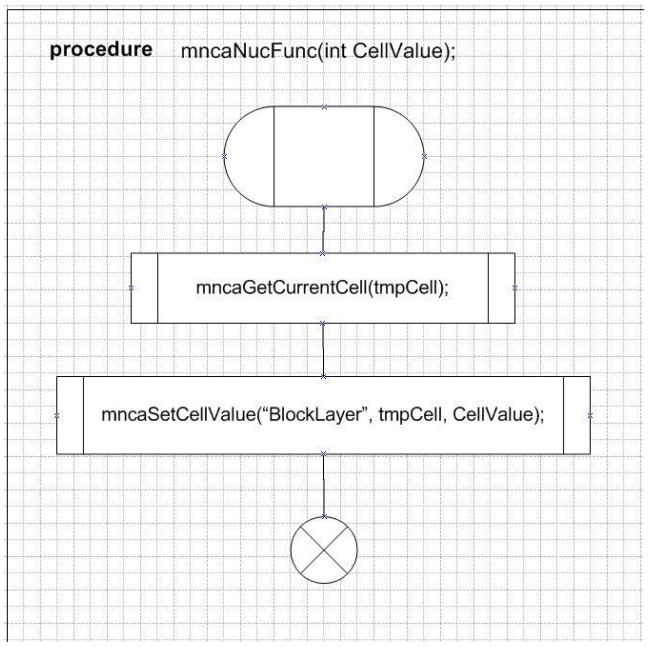

procedure mncaNucFunc(int CellValue);
mncaGetCurrentCell(tmpCell);
mncaSetCellValue("BlockLayer", tmpCell, CellValue);

Figure 13: Nucleus function, defining the cells that must be modified due to the execution of the cellular automata evolution function.

8. EXECUTING THE MODEL

Since cellular automata frequently are used to perform environmental simulations, is needed to define a method to load this kind of information in the model. Since in a m:n-CA^k cellular automata different layers can be used, different layers must be defined to load the initial automata configuration (one for each one of the layers of the automata). In our example, since only one layer is used the files used are:

- Layer.doc: File containing the description of the raster file.
- Layer.img: File containing the data of the raster file. This data is used to fill the cellular automata initial state. In our case the file is containing 0's.

At the end of the simulation we obtain a new layer (Result_Layer.img) that contains the new values for the layer. In our Fibonacci example the file have the next output:

```
0\n1\n1\n2\n3\n5\n8\n13\n21\n34\n55\n89\n
(…)
```

The structures of these files are based on the IDRISI32 file format[14]. In brief the initial conditions of the cellular automata can be defined using GIS data that follows IDRISI32 format. The output is a new file that follows IDRISI32 format.

The model is represented using Microsoft Visio®. From this representation an XML is obtained (thanks a plug-in). This XML is loaded in SDLPS who performs the simulation. We are using XML file instead the SDL-PR representation because XML offers different advantages to manage the data. Also allows the incorporation of blocs and metadata inside the XML model representation. This metadata is useful to add information about the graphical representation that we want that each one of the different SDL agents have.

9. CONCLUSIONS AND FUTURE WORK

This paper proposes a solution to represent the behavior of cellular automata graphically using the Specification and Description Language.

To do this we propose two solutions to two existing problems. First, we had shown how to deal with time in SDL. Second how to represent cellular automata using the language. To do this first an extension of the common cellular automata is presented. This extension allows the definition of several functions that helps in the description of its behavior.

From these extensions we had shown how a Fibonacci model can be represented using SDL.

The graphical representation of the cellular automata behavior helps in the understanding of its behavior. In this kind of models, where usually are formed by multidisciplinary individuals, these graphical tool can be very valuable.

The future work is focused in the implementation of all this extensions in a way that allows a distributed execution of the automata with a dynamic assignation of the existing resources. Also we are developing some different models using this approximation to represent environmental phenomena, like slap avalanches [15] or wildfires.

10. REFERENCES

[1] Brade, D. (2000). Enhancing modeling and simulation accreditation by structuring verification and validation results. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Ed.), *Winter Simulation Conference.*

[2] De Lara, J., & Vangheluwe, H. (2002). ATOM³: A Tool for Multi-formalism Modelling and Meta-modelling. *4th International Conference on Enterprise Information Systems ICEIS 2002.*

[3] Praehofer, H., & Pree, D. (1993). Visual modeling of DEVS-based multiformalism systems based on higraphs. In G. W. Evans, M. Mollaghasemi, E. C. Russell, & W. E. Biles (Ed.), *Winter Simulation Conference*, (pp. 595-603). Los Angeles, California, United States.

[4] Wainer, G., & Chen, W. (2003, November). A framework for remote execution and visualization of Cell-DEVS models. *Simulation: Transactions of the Society for Modeling and Simulation International* , 626-647.

[5] Telecommunication standardization sector of ITU. (1999). *Specification and Description Language (SDL).* Retrieved April 2008, from Series Z: Languages and general software aspects for telecommunication systems.: http://www.itu.int/ITU-T/studygroups/com17/languages/index.html

[6] CINDERELLA SOFTWARE. (2007). *Cinderella SDL*. Retrieved 03 31, 2009, from http://www.cinderella.dk

[7] IBM. (2009). *TELELOGIC*. Retrieved 03 31, 2009, from http://www.telelogic.com/

[8] Fonseca i Casas, P. (2008). SDL distributed simulator. *Winter Simulation Conference 2008.* Miami: INFORMS.

[9] Emmeche, C. (1998). *Vida Simulada en el ordenador*. Barcelona, Catalunya: Gedisa.

[10] Fonseca i Casas, P., & Casanovas, J. (2005). Simplifying GIS data use inside discrete event simulation model through m_n-AC cellular automaton. *Proceedings ESS 2005.*

[11] Arregui Fernandez, J. (1988). *Topología*. UNED.

[12] Fonseca i Casas, P., Casanvas, J., & Montero, J. (2004). GIS and simulation system integration in a virtual reality environment. *Proceedings GISRUK 2004*, (pp. 403-408).

[13] Fujimoto, R. M. (2001). Parallel simulation: parallel and distributed simulation systems. *Winter Simulation Conference*, (pp. 147-157).

[14] Clark Labs. (2009). *IDRISI*. Retrieved 11 26, 2009, from IDRISI: http://www.idrisi.com/

[15] Fonseca i Casas, P., & Rodríguez Fontoba, S. (2007). Using GIS data in a m:n-ACk cellular automaton to perform an avalanche simulation. *Geographical Information Science Research UK Conference 2007.* National University of Ireland Maynooth.

[16] Law, A. M., & Kelton, W. D. (2000). *Simulation Modeling and Analysis.* McGraw-Hill.

[17] Guasch, A., Piera, M. À., Casanovas, J., & Figueras, J. (2002). *Modelado y simulación.* Barcelona, Catalunya/Spain: Edicions UPC.

[18] Fishman, G. S. (2001). *Discrete-Event Simulation: Modeling, Programming and Analysis.* Berlin: Springer-Verlag.

[19] Bozga, M., Graf, S., Mounier, L., Kerbrat, A., Ober, I., & Vincent, D. (2000). SDL for Real-Time: What Is Missing ? *SAM'2000.* Grenoble, France.

[20] Bozga, M., Graf, S., Mounier, L., Ober, I., Roux, J.-L., & Vincent, D. (2001). Timed Extensions for SDL. *Proceedings of SDL-Forum'01.* Copenhagen, Denmark.

[21] Doldi, L. (2003). *Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis.* John Wiley & Sons, Inc.

[22] Banks, J., & Gibson, R. (1997, February). Simulation modeling: some programming required. *IIE Solutions* , 26-31.