Aplicación del formalismo Cell-DEVS usando la herramienta N-CD++

Daniel A. Rodríguez
Drodrigu@dc.uba.ar

Gabriel A. Wainer
gabrielw@dc.uba.ar

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
(1428) Pabellón I. Ciudad Universitaria.
Buenos Aires. Argentina.

RESUMEN

El presente trabajo describe las extensiones realizadas a una herramienta utilizada para el estudio, modelado y simulación de modelos celulares n-dimensionales, respetando los paradigmas DEVS y Cell-DEVS. La misma posibilita, mediante el uso de un lenguaje de especificación de alto nivel, la construcción de sistemas basados en celdas. Además, permite lograr una disminución en los tiempos de desarrollo, chequeo y mantenimiento de los componentes, posibilitando incluso la reusabilidad de los mismos. Se muestra la utilidad del formalismo y de la herramienta por medio de diversos ejemplos de aplicación.

Palabras Claves: Simulación, Autómata Celular, DEVS, Cell-DEVS.

1. INTRODUCCIÓN

Existen sistemas complejos para los cuales se hace muy difícil, y en ciertos casos imposible, hallar soluciones analíticas o heurísticas con resultados satisfactorios. Para el estudio de este tipo de sistemas se ha difundido el uso de metodologías y herramientas de simulación.

Actualmente, para gran variedad de aplicaciones complejas se usan modelos y simulación. En algunos de estos sistemas, las variables son discretas a tiempo continuo. Tales sistemas reciben el nombre de ***Sistemas Dinámicos de Eventos Discretos*** (**DEDS** – Discrete Events Dynamic Systems). Un paradigma de simulación para DEDS asume que el sistema simulado sólo cambia de estado en puntos discretos del tiempo, ante la ocurrencia de un evento.

En [Zei76] se propuso un formalismo conocido como DEVS (Discrete EVents systems Specifications) que permite la construcción jerárquica de modelos de eventos discretos. Esto favorece la creación de los mismos, ya que pueden ser considerados como nuevos componentes para ser usados en la creación de otros modelos, reduciendo así los tiempos de desarrollo y prueba.

Un modelo DEVS se construye sobre la base de un conjunto de modelos básicos llamados **Atómicos**, que se combinan para formar modelos **Acoplados** y un conjunto de relaciones que indican como los modelos están conectados en forma jerárquica.

Los ***Autómatas Celulares*** son un formalismo que permiten definir un conjunto infinito n-dimensional de celdas con igual comportamiento. El vecindario de una celda es un conjunto de celdas cercanas a la misma que permite actualizar su estado en forma independiente a las demás [Wol86].

Cell-DEVS [WG98] es un formalismo que se basa en DEVS, extendiéndolo para poder implementar modelos celulares. En el paradigma Cell-DEVS cada celda se define como un modelo atómico con un conjunto de estados para los valores del vecindario, un comportamiento y una demora de actualización que puede ser de transporte ó inercial. Un modelo acoplado que incluya a un conjunto de estas celdas formará así a un modelo celular.

Los modelos atómicos Cell-DEVS pueden describirse formalmente como:

$$CA = < X, Y, I, demora, S, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

donde:

X es el conjunto de eventos de entrada;
Y es el conjunto de eventos de salida;
I representa la definición de la interfaz del modelo;
La **demora** puede ser *de transporte* o *inercial*;
S es el conjunto de estados de la celda;
N es el conjunto de valores de los vecinos;
d es la demora de la celda;
δ_{int} es la función de transición interna;
δ_{ext} es la función de transición externa;
τ es la función de cálculo local;
λ es la función de salida; y
D es la función de duración de vida del estado.

La especificación permite definir una celda como un modelo atómico modular. Un modelo Cell-DEVS acoplado n-dimensional puede definirse como:

$$CC = < Xlist, Ylist, I, X, Y, k, \{t_1,...,t_k\}, \eta, N, C, B, Z, select >$$

donde:

Ylist es la lista de acoplamiento externo;
Xlist es la lista de acoplamiento interno;
I es la interfaz de comunicación con los modelos externos;
k es la dimensión del espacio de celdas;
$\{t_1,..., t_k\}$ es la cantidad de celdas en cada una de las dimensiones;
η es el tamaño del vecindario;
X es el conjunto de eventos externos de entrada;
Y es el conjunto de eventos externos de salida;
N es la definición del vecindario;
C es el conjunto de celdas atómicas que conforman al modelo acoplado;
B es el conjunto de celdas que representan el borde del modelo celular;
Z define el acoplamiento interno (conexión de puestos de E/S entre celdas); y
select es la función de selección de una celda inminente ante eventos simultáneos.

CD++ [BBW98b][RW99] es una herramienta que permite implementar los conceptos teóricos especificados anteriormente, y que soporta la creación de autómatas celulares bidimensionales permitiendo, además, su combinación con otros modelos DEVS. Fue construida usando C++ definiendo una jerarquía de clases para los modelos atómicos que extiende a la definida en **GAD** [BBW98a], la cual no incluía modelos celulares. Un lenguaje de especificación permite la creación de modelos acoplados, la configuración inicial para los modelos atómicos, y la creación de eventos externos para poder ser usados durante la simulación, así como también permite describir el comportamiento de cada celda de un modelo celular.

El objetivo de este trabajo es presentar la herramienta N-CD++ utilizada para la simulación de modelos Cell-DEVS, y que esta basada en CD++, extendiéndola para soportar modelos celulares n-dimensionales. Posteriormente se ejemplificará la definición de modelos usando la citada herramienta.

2. SOPORTE DE MODELOS N-DIMENSIONALES

Los problemas reales que implican el uso de autómatas celulares generalmente deben representarse usando modelos en dos o tres dimensiones. A esto puede sumársele una serie de problemas teóricos donde es posible el uso de cuatro o más dimensiones para poder dar una respuesta a los mismos. La versión original de la herramienta sólo permite definir autómatas celulares bidimensionales y lineales, aunque estos últimos sean un caso particular de los primeros, ya que se definen mediante la utilización de un autómata bidimensional especificando una dimensión (*d*, 1), donde *d* es el número de celdas. Esto restringe los modelos a ser implementados, limitando así el uso de la herramienta en problemas más generales. Ante esta situación se decidió la expansión de la herramienta para posibilitar la creación de autómatas celulares n-dimensionales.

Según la definición de autómatas celulares, estos pueden ser o no toroidales. Este concepto sigue siendo válido incluso con las extensiones realizadas: si un autómata celular tiene dimensión $(d_1, d_2, ..., d_n)$, entonces la referencia a una celda $(x_1, x_2, ..., x_n)$ se corresponde con la celda $(x_1 \bmod d_1, x_2 \bmod d_2, ..., x_n \bmod d_n)$.

CD++ almacena internamente el espacio celular de dimensión (d_1, d_2) sobre un arreglo de $d_1 \cdot d_2$ elementos, donde el elemento (x_1, x_2), con $x_i \in [0, d_i - 1]$, ocupa la posición $x_1 + x_2 \cdot d_1$. Análogamente, N-CD++ usa un arreglo de $\Pi_{i=1...n} d_i$ elementos para almacenar el autómata celular de dimensión $(d_1, d_2, ..., d_n)$, y en este caso el elemento $(x_1, x_2, ..., x_n)$ ocupa la posición $\Sigma_{i=1...n} x_i \cdot (\Pi_{k=1...i-1} d_k)$.

Asimismo, en la nueva versión, y debido a que la dimensión del espacio celular depende del modelo que se este interpretando, se debió crear un tipo de datos, al cual se lo llamó *nTupla*, y que almacena valores numéricos enteros en un orden arbitrario. De esta forma es posible almacenar una posición dentro del espacio celular de dimensión *n*, usando una *nTupla* que almacene exactamente *n* valores.

La definición de cada modelo celular se realiza mediante un lenguaje de especificación que permite definir la dimensión del mismo, el valor inicial de cada celda y el vecindario utilizado. Además, permite la definición del comportamiento de cada celda utilizando reglas definidas según la gramática mostrada en el *Apéndice A*.

En CD++ el vecindario de una celda puede estar compuesto solo por celdas del espacio celular adyacentes a la celda en cuestión. De esta forma, y debido a que se utiliza un espacio celular de dos dimensiones, el vecindario de una celda puede tener como máximo 9 elementos (considerando a la celda como integrante del vecindario). N-CD++ permite la definición del vecindario de una celda donde los elementos que lo componen no

necesariamente deben encontrarse adyacentes a la celda en cuestión. Además, este vecindario puede tener igual o menor dimensión que la usada por el autómata celular.

Por otra parte, se permite la definición de *zonas*. Una *zona* es un conjunto de celdas contenidas en el área determinada por el rango $\{celda_1..celda_2\}$, como se muestra en la Figura 1, asociado a un conjunto de reglas. De esta forma, distintas zonas dentro de un mismo modelo celular pueden tener distinto comportamiento. Este concepto se ha mantenido en N-CD++, permitiendo definir rangos cuya dimensión puede ser igual e incluso menor a la dimensión del autómata. En este caso, una zona definida por el rango $\{(x_1, x_2, ..., x_n)...(y_1, y_2, ..., y_n)\}$ es el conjunto de celdas $(t_1, t_2, ..., t_n)$ del autómata celular, tales que $\min(x_i, y_i) \leq t_i \leq \max(x_i, y_i)$, $\forall\ i \in [1, n]$.

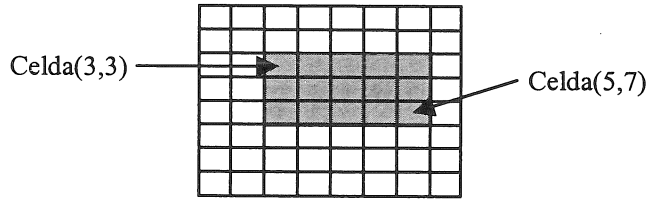

Figura 1 - Definición de la zona { (3,3)...(5,7) }

3. EJEMPLOS DE APLICACIÓN

3.1. Variante del *Juego de la Vida*

Pasaremos a ejemplificar el uso de la herramienta. En primer caso se muestra una variación del *Juego de la Vida* [Gar70], donde inicialmente se cuenta con una población compuesta por seres representados por el valor 1, dispersos en un área de 7 x 7 x 3 celdas, de las cuales algunas contienen el valor 0 indicando la ausencia de cualquier tipo de ser. El nacimiento de un nuevo ser se produce cuando la celda tiene diez o más seres vivos como vecinos. Por otra parte, un ser permanecerá vivo mientras que su vecindario contenga 8 ó 10 seres vivos. En cualquier otro caso la celda contendrá el valor 0, debido a que ya estaba vacía o a que el ser que la ocupaba ha muerto.

Para este ejemplo se usará un vecindario de dimensión 3 x 3 x 3, según se muestra en la siguiente Figura 2.

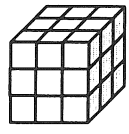

Figura 2 - Vecindario para la variante del Juego de la Vida

En la Figura 3 se muestra la descripción del modelo en el lenguaje provisto por la herramienta, mientras que en la Figura 4 se muestra el estado inicial de las celdas.

```
01      [top]
02      components : 3d-life
03
04      [3d-life]
05      type : cell
06      dim : (7,7,3)
07      delay : transport
08      defaultDelayTime  : 100
09      border : wrapped
10      neighbors : 3d-life(-1,-1,-1) 3d-life(-1,0,-1) 3d-life(-1,1,-1)
11      neighbors : 3d-life(0,-1,-1)  3d-life(0,0,-1)  3d-life(0,1,-1)
12      neighbors : 3d-life(1,-1,-1)  3d-life(1,0,-1)  3d-life(1,1,-1)
13      neighbors : 3d-life(-1,-1,0)  3d-life(-1,0,0)  3d-life(-1,1,0)
14      neighbors : 3d-life(0,-1,0)   3d-life(0,0,0)   3d-life(0,1,0)
15      neighbors : 3d-life(1,-1,0)   3d-life(1,0,0)   3d-life(1,1,0)
16      neighbors : 3d-life(-1,-1,1)  3d-life(-1,0,1)  3d-life(-1,1,1)
17      neighbors : 3d-life(0,-1,1)   3d-life(0,0,1)   3d-life(0,1,1)
18      neighbors : 3d-life(1,-1,1)   3d-life(1,0,1)   3d-life(1,1,1)
19      initialvalue : 0
20      initialCellsValue : 3d-life.val
21      localtransition : 3d-life-rule
22
23      [3d-life-rule]
24      rule : 1 100 { (0,0,0) = 1 and (truecount = 8 or truecount = 10) }
25      rule : 1 100 { (0,0,0) = 0 and truecount >= 10 }
26      rule : 0 100 { t }
```

Figura 3 - Descripción de la variante del *Juego de la Vida*

```
(0,0,0) = 1     (1,5,1) = 1     (2,6,1) = 1     (4,5,0) = 1     (5,5,2) = 1
(0,0,2) = 1     (1,6,0) = 1     (3,2,1) = 1     (4,5,2) = 1     (5,6,0) = 1
(1,0,0) = 1     (1,6,1) = 1     (3,5,1) = 1     (4,6,0) = 1     (6,0,0) = 1
(1,0,1) = 1     (2,1,2) = 1     (3,5,2) = 1     (4,6,2) = 1     (6,1,1) = 1
(1,1,1) = 1     (2,1,0) = 1     (3,6,1) = 1     (5,1,2) = 1     (6,1,2) = 1
(1,2,0) = 1     (2,3,1) = 1     (3,6,2) = 1     (5,2,0) = 1     (6,3,0) = 1
(1,2,2) = 1     (2,3,2) = 1     (4,1,2) = 1     (5,2,2) = 1     (6,3,2) = 1
(1,3,2) = 1     (2,4,1) = 1     (4,2,0) = 1     (5,3,0) = 1     (6,4,2) = 1
(1,4,2) = 1     (2,4,2) = 1     (4,2,1) = 1     (5,3,1) = 1     (6,5,1) = 1
(1,5,0) = 1     (2,5,0) = 1     (4,4,1) = 1     (5,5,1) = 1     (6,6,0) = 1
                                                                (6,6,2) = 1
```

Figura 4 - Estado inicial de las celdas para la variante del Juego de la Vida

Entre las líneas 1 a 2 se define el modelo acoplado de mayor nivel, compuesto en este caso por el modelo 3d-*life*. El resto de las líneas define a este modelo.

Entre las líneas 4 a 19 se define la dimensión del espacio de celdas, el tipo de demora y el vecindario. En la línea 20 se define el archivo que contiene los valores iniciales para las celdas, cuyo contenido es mostrado en la Figura 4. En la línea 21 se especifica el nombre de la función de transición usada para calcular el valor de las celdas en cada etapa de la simulación. Esta función se define en las líneas 23 a 26.

Los resultados arrojados por la herramienta es el siguiente:

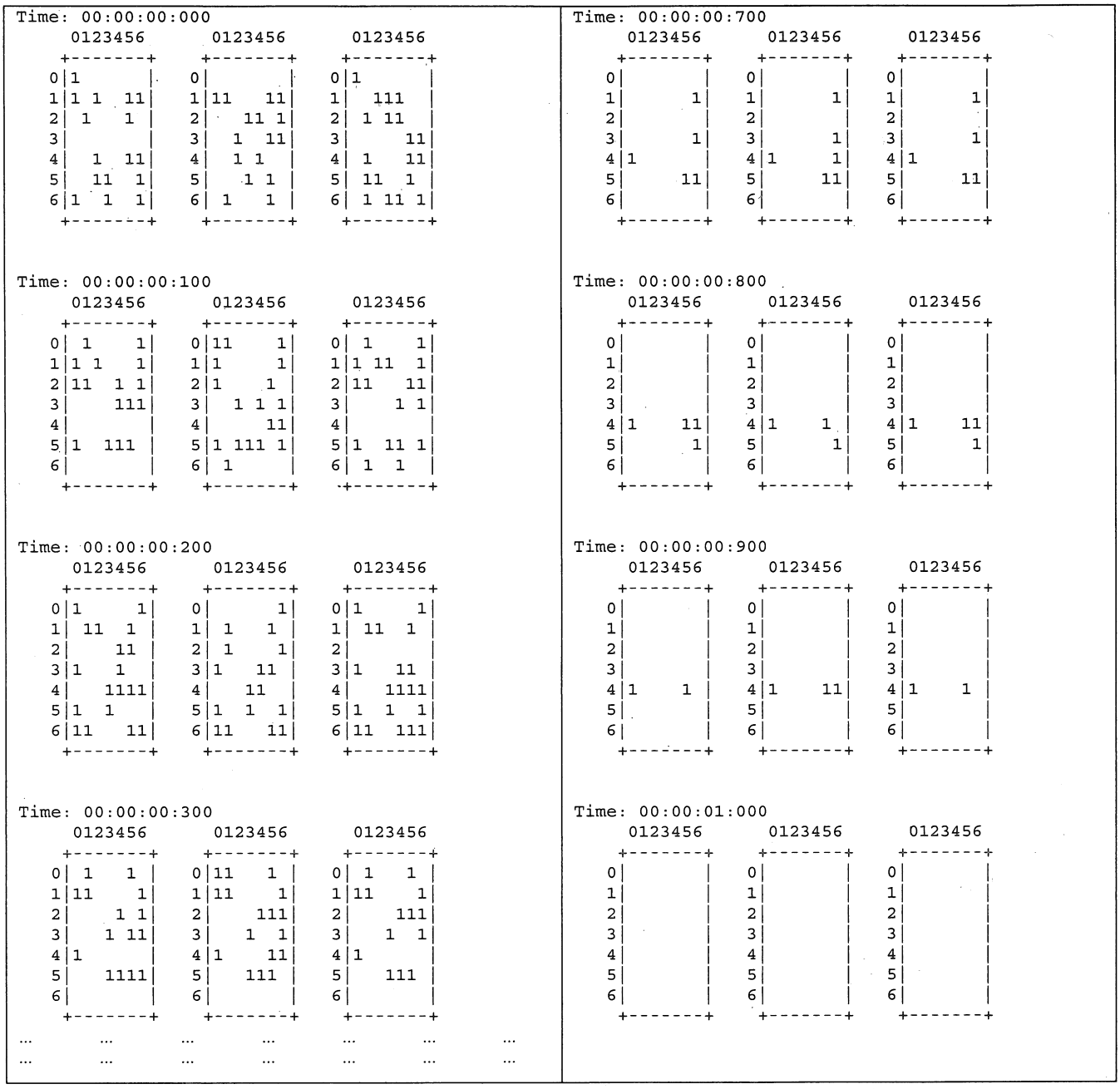

Figura 5 - Resultados Arrojados para la variante del *Juego de la Vida*

Aquí puede apreciarse que inicialmente existen seres dispersos por todo el espacio celular, pero su distribución hace que la población no sea estable, tendiendo a disminuir en cantidad, lo que finalmente sucede en el tiempo 00:00:01:000 con la extinción de los mismos.

3.2. Difusión del Calor

En este ejemplo se cuenta con un ambiente representada por un autómata celular, donde cada celda contiene una temperatura. En cada etapa de la simulación, la temperatura de la celda se calcula como el promedio de los valores de la vecindad, cuya forma se muestra en la Figura 6.

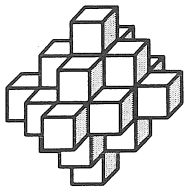

Figura 6 - Vecindario para el ejemplo de Difusión del Calor

Además, existe un generador de calor que se conecta a las celdas (2,2,1) y (3,3,0), el cual permite la creación de temperaturas en el rango [24, 80] grados centígrados, con distribución uniforme. Por otra parte, se cuenta con un generador de frío, que permite crear valores en el rango [-45, 10] también con distribución uniforme, y se conecta a las celdas (1,3,3) y (3,3,2). Ambos generadores crean valores luego de transcurridos x segundos, donde x sigue una distribución exponencial con media 40 segundos. El esquema del acoplamiento de los modelos se muestra en la Figura 7.

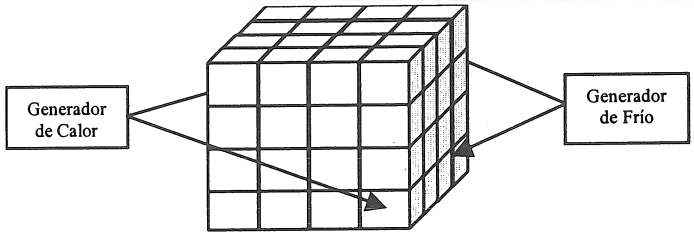

Figura 7 - Esquema del acoplamiento del modelo de difusión de calor

La definición del modelo usando el lenguaje provisto por la herramienta se muestra en la Figura 8. Entre las líneas 1 y 4 se define el modelo con mayor nivel de abstracción, junto a los modelos que los integran. Además, se muestra el acoplamiento entre los mismos. Entre las líneas 6 y 32 se define el modelo que representa el ambiente, compuesto por un autómata celular de 10 x 10 x 4 celdas, donde inicialmente todas tienen una temperatura de 24° C (según lo definido en la línea 22). Entre las líneas 34 y 38 se define la función que calcula el valor de estado de las celdas como el promedio de los valores del vecindario. En las líneas 40 y 41 se define la función que crea una temperatura en el rango [24, 80] con distribución uniforme, al ingresar un mensaje por el puerto In de las celdas (2,2,1) y (3,3,0) como se determinó en las líneas 24 y 25. Análogamente, en las líneas 43 y 44 se define la función para crear temperaturas en el rango [-45, 10] con distribución uniforme, al ingresar un mensaje por el puerto In de las celdas (1,3,3) y (3,3,2). Finalmente, entre las líneas 46 y 56 se definen los generadores de frío y calor, que generan valores cada x segundos, donde x sigue una distribución exponencial con media 40 segundos.

El resumen de los resultados de salida generados por la herramienta se muestran en la Figura 9. En el tiempo 00:00:01:000 los generadores de frío y calor producen cambios en las celdas a las cuales están conectadas. Esto conduce a que los estados de los vecinos de estas celdas cambien en 00:00:02:000, y en las sucesivas etapas se producirá un cambio en el resto de las celdas. En el tiempo 00:00:15:738, el generador de frío producirá un cambio en el estado de las celdas (1,3,3) y (3,3,2), estableciéndose los valores -24 y –43.1 respectivamente, lo que producirá futuros cambios de estados en los vecinos de las mismas.

```
01    [top]
02    components : superficie generadorCalor@Generator generadorFrio@Generator
03    link : out@generadorCalor inputCalor@superficie
04    link : out@generadorFrio  inputFrio@superficie
05
06    [superficie]
07    type : cell
08    dim : (4, 4, 4)
09    delay : transport
10    defaultDelayTime  : 100
11    border : wrapped
12    neighbors :                          superficie(-1,0,-1)
13    neighbors : superficie(0,-1,-1)  superficie(0,0,-1)  superficie(0,1,-1)
14    neighbors :                          superficie(1,0,-1)
15    neighbors : superficie(-1,-1,0)  superficie(-1,0,0)  superficie(-1,1,0)
16    neighbors : superficie(0,-1,0)   superficie(0,0,0)   superficie(0,1,0)
17    neighbors : superficie(1,-1,0)   superficie(1,0,0)   superficie(1,1,0)
18    neighbors :                          superficie(-1,0,1)
19    neighbors : superficie(0,-1,1)   superficie(0,0,1)   superficie(0,1,1)
20    neighbors :                          superficie(1,0,1)
21    neighbors : superficie(0,0,-2)   superficie(0,0,2)   superficie(0,2,0)
21    neighbors : superficie(0,-2,0)   superficie(2,0,0)   superficie(-2,0,0)
22    initialvalue : 24
23    in : inputCalor inputFrio
24    link : inputCalor in@superficie(3,3,0)
25    link : inputCalor in@superficie(2,2,1)
26    link : inputFrio  in@superficie(3,3,2)
27    link : inputFrio  in@superficie(1,3,3)
28    localtransition : calor-rule
29    portInTransition : in@superficie(3,3,0)  setCalor
30    portInTransition : in@superficie(2,2,1)  setCalor
31    portInTransition : in@superficie(3,3,2)  setFrio
32    portInTransition : in@superficie(1,3,3)  setFrio
33
34    [calor-rule]
35    rule : { ( (-1,0,-1) + (0,-1,-1) + (0,0,-1) + (0,1,-1) + (1,0,-1) + (-1,-1,0) +
36           (-1,0,0) + (-1,1,0)  + (0,-1,0) + (0,0,0) + (0,1,0) + (1,-1,0) + (1,0,0) +
37           (1,1,0) + (-1,0,1) + (0,-1,1) + (0,0,1) + (0,1,1) + (1,0,1) + (0,0,-2) +
38           (0,0,2) + (0,2,0) + (0,-2,0) + (2,0,0) + (-2,0,0) ) / 25 } 1000 { t }
39
40    [setCalor]
41    rule : { uniform(24,80) } 1000 { t }
42
43    [setFrio]
44    rule : { uniform(-45,10) } 1000 { t }
45
46    [generadorCalor]
47    distribution : exponential
48    mean : 10
49    initial : 1
50    increment : 0
51
52    [generadorFrio]
53    distribution : exponential
54    mean : 10
55    initial : 1
56    increment : 0
```

Figura 8 - Definición del modelo de difusión del calor

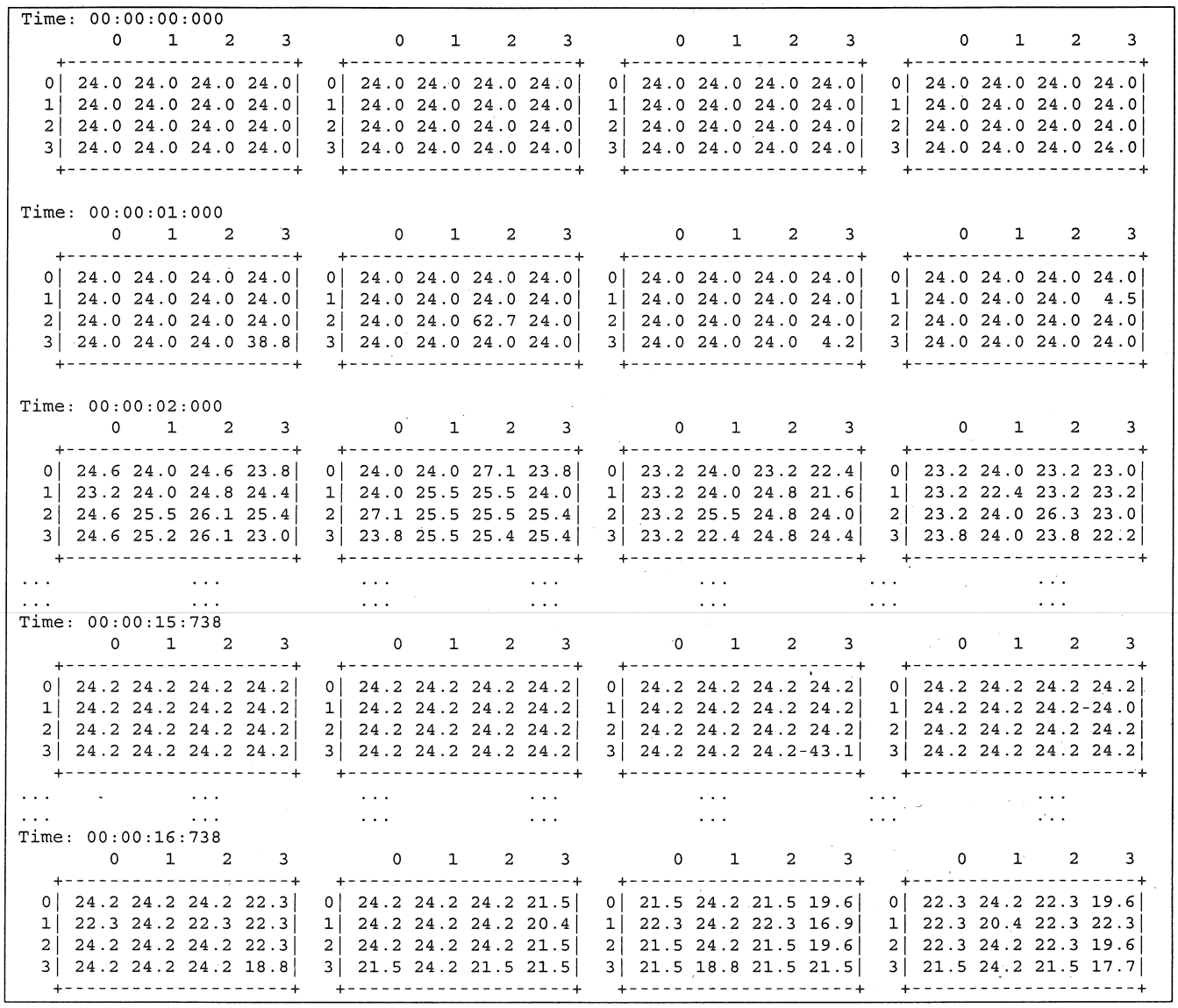

Figura 9 – Resumen de los resultados obtenidos durante la simulación de difusión del calor

4. CONCLUSIONES

En este trabajo se presentó una extensión a la herramienta CD++ utilizada para el modelado y simulación de modelos DEVS permitiendo, además, la creación de modelos celulares. Este formalismo posibilita la construcción de modelos jerárquicos, basándose en modelos más simples y de menor tamaño, lo que facilita las etapas de desarrollo, chequeo y mantenimiento de los mismos, y posibilitando la reusabilidad de sus componentes.

Las extensiones desarrolladas a la herramienta permiten, mediante la utilización de un lenguaje sencillo y conciso, representar nuevos modelos los cuales en la versión original eran imposibles de simular.

Para concluir, cabe considerar que como la herramienta fue construida usando una versión estándar de C++ esto favoreció la portabilidad de la misma a distintas plataformas sin costo adicional. Hasta el momento

CD++ fue testeado en forma exitosa en Windows 95 / NT, Linux, AIX, HP-UX y Sun, lo que creó un incentivo en los usuarios a realizar simulaciones en su ámbito habitual para luego poder realizar estudios más profundos con máquinas con mayor poder de cómputo.

REFERENCIAS

[BBW98a] BARYLKO, A.; BEYOGLONIAN, J.; WAINER G. *"GAD: A General Application Tool for DEVS Modeling and Simulation"*. En Proceedings of IASTED International Conference on Applied Modelling and Simulation, Hawai, USA, 1998.

[BBW98b] BARYLKO, A.; BEYOGLONIAN, J.; WAINER G. "CD++: Una Herramienta de Implementación de Modelos Cell-DEVS Binarios". Memorias de la XXIV Conferencia Latinoamericana de Informática. Quito, Ecuador, 1998.

[Gar70] GARDNER, M. "The Fantastic Combinations of John Conway's New Solitaire Game 'Life' ". Scientific American, 23 (4), 1970, pp. 120-123.

[RW99] RODRÍGUEZ, D.; WAINER G. "Redefinition of a specification language for Cell-DEVS models". Enviado a Fifth International Conference on Information Systems Analysis and Synthesis (ISAS'99). Orlando, USA, 1999.

[Wol86] WOLFRAM, S. "Theory and Applications of Cellular Automata". Vol. 1, Advances Series on Complex Systems. World Scientific, Singapore, 1986.

[WG98] WAINER, G.; GIAMBIASI, N. "Specification, Modelling and Simulation of Timed Cell-DEVS Models". Technical Report TR-98-006, Departamento de Computación, FCEyN/UBA. Enviado para su publicación a ACM Transactions on Modelling and Simulation. 1998.

[Zei76] ZEIGLER, B. *"Theory of Modelling and Simulation"*. Wiley, N.Y. 1976.

APÉNDICE A. LENGUAJE DE ESPECIFICACIÓN DE N-CD++

La sintaxis del lenguaje usado por N-CD++ para la especificación del comportamiento de los modelos celulares atómicos puede definirse con la siguiente BNF:

```
RULELIST   = RULE  |  RULELIST

RULE       = Rule : RESULT RESULT { BOOLEXP }

RESULT     = CONSTANT  |  { REALEXP }

BOOLEXP    = BOOL  |  (BOOLEXP)  |  REALRELEXP  |  not BOOLEXP  |  BOOLEXP OP_BOOL BOOLEXP

OP_BOOL    = and  |  or  |  xor  |  imp  |  eqv

REALRELEXP = REALEXP OP_REL REALEXP  |  COND_REAL_FUNC(REALEXP)

REALEXP    = IDREF  |  (REALEXP)  |  REALEXP OPER REALEXP

IDREF      = CELLREF  |  CONSTANT  |  FUNCTION  |  portValue(PORTNAME)

CONSTANT   = INT  |  REAL  |  CONSTFUNC  |  ?

FUNCTION   = UNARY_FUNC(REALEXP) | WITHOUT_PARAM_FUNC |  BINARY_FUNC(REALEXP, REALEXP)  |
             if(BOOLEXP, REALEXP, REALEXP)  |  ifu(BOOLEXP, REALEXP, REALEXP, REALEXP)

CELLREF    = (INT, INT RESTO_TUPLA

RESTO_TUPLA = , INT RESTO_TUPLA  |  )

BOOL       = t  |  f  | ?

OP_REL     = !=  |  =  |  >  |  <  |  >=  |  <=

OPER       = +  |  -  |  *  |  /

INT        = [SIGN] DIGIT {DIGIT}

REAL       = INT  |  [SIGN] {DIGIT}.DIGIT {DIGIT}

SIGN       = +  |  -

DIGIT      = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

PORTNAME   = thisPort  |  STRING

STRING     = LETTER {LETTER}

LETTER     = a | b | c |...| z | A | B | C |...| Z

CONSTFUNC  = pi | e | inf | grav | accel | light | planck | avogadro | faraday | rydberg |
             euler_gamma| bohr_radius | boltzmann | bohr_magneton | golden | catalan | amu |
             electron_charge | ideal_gas | pem | stefan_boltzmann | proton_mass |
             electron_mass | neutron_mass

WITHOUT_PARAM_FUNC = truecount | falsecount | undefcount | time | random | randomSign

UNARY_FUNC = abs | acos | acosh | asin | asinh | atan | atanh | cos | sec | sech | exp | cosh | fact |
             fractional | ln | log | round | cotan | cosec | cosech | sign | sin | sinh | statecount |
             sqrt | tan | tanh | trunc | truncUpper | poisson | exponential | randInt | chi | asec |
             acotan | asech | acosech | nextPrime | radToDeg | degToRad | nth_prime | acotanh | CtoF |
             CtoK | KtoC | KtoF | FtoC | FtoK

BINARY_FUNC = comb | logn | max | min | power | remainder | root | beta | gamma | lcm | gcd | normal |
              f | uniform | binomial | rectToPolar_r | rectToPolar_angle | polarToRect_x |
              polarToRect_y | hip
COND_REAL_FUNC = even | odd | isInt | isPrime | isUndefined
```