Ciencias de la Computación - Tesis de Licenciatura

Departamento de Computación
Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires
2003

CD++ - Analysis of Quantized Continuous Cell-DEVS

Author
Leandro San Miguel

Director
Dr. Gabriel Wainer

Abstract

Cell-DEVS is a formalism intended to model cell spaces. It describes cellular models using timing delay constructions, allowing simple definition of complex timing. Complex timing models generates million of messages, big data files and long processing time. This work shows the quantification mechanisms and the analyzed results of the different model quantification (with complex functions) in CD++ in order to simplify the models, reducing the quantity of messages generated and processing time.
CD++ models were revised, implemented and adapted to analyze complex local computation functions with different quantum types and techniques. Quantum standard, quantum Hysteresis, Dynamic Quantums, Adaptive Quantization and Generalized Discrete Event Simulation models are implemented and analyzed with three complex models. These three models are: The Heart tissue model, The Watershed model and The Flow Injection Analysis model.

1 Introduction

Simulation is a powerful tool for analyzing complex systems. The simulation process starts with a problem to solve or understand from the observation of a real system. Entities are identified, and an abstract representation, a model, is constructed. The execution of the model is done by a simulator, which consists of a computer system that executes the model's instructions to generate its behavior. To complete the cycle, the results obtained are compared to those of the real system for model validation. It is often the case that a modeler is only interested in a few aspects of the real system. In such a case, an experimental frame captures the modeler's objectives and defines the scope of the model.

At present exist a quite number of simulation techniques and paradigms. Among these, the DEVS formalism [1] provides a framework for the construction of hierarchical models in a modular manner, allowing for model reuse and reducing development time and testing. In DEVS a model is specified as a black box with a state and a duration for that state. When the duration time for the state expires, an output event is sent, an internal transition takes place and the model changes its current state. A change of state can also occur when an external event is received. Then, describing the set of states a model goes through, the internal and external transition functions, the output function and the state duration function define a complete model. DEVS models can be put together by linking the outputs of a model to inputs of other models to form coupled models. Models made out of only one component are called atomic.

DEVS not only proposes a framework for model construction, but also defines an abstract simulation mechanism that is independent of the model itself. This mechanism is high-level description of how the simulation of DEVS models should be executed by a simulator. Two kinds of simulators are defined, one for atomic and another one for coupled models. These simulators progress through the simulation by exchanging messages as described by an abstract simulation mechanism.

Timed Cell-DEVS [2] is a formalism based on DEVS for the simulation of cellular models. A cellular automaton is a lattice of cells, each of which has a value and a local rule that defines how to obtain a new value based on the current state of the cell and the values of neighboring cells. Cells are synchronously updated at the same time. Timed Cell-DEVS defines a cell as a DEVS model and a cellular automaton as a coupled model. In Timed Cell-DEVS each cell defines its own update delay.

CD++ is a tool for the simulation of DEVS and Cell-DEVS models which has been used to simulate a variety of models including: traffic, forest fires, ants and physical phenomena. Simple models were easily handled by the tool, but the execution of complex models results in big data files and high processing time. That is as a result of the transitions and output functions exchanging thousands (and millions, depending on the model) of messages, which is the way the CD++ tool implements the transitions status and values changes). To minimize the message generation (and consequently the processing time and result data files) a theory of quantized models was developed [4].
When using quantized models, a cell's state value will be only informed to its neighbors if its difference with the previous value is greater than a given quantum. This operation reduces substantially the frequency of message updates, while potentially incurring into error. Different quantum techniques are implemented and the result of applying these techniques can be different depending on the model behavior and the quantum value used.

The aim of this work is to analyze the error, time and number of messages while applying different quantum techniques to different complex models. When quantifying models, some implementation details were revised in order to enhance the simulation and the analysis.

Several quantum mechanisms were implemented, analyzed and applied to the three-implemented models with a local computation function providing high variation and extreme sensitivity to the quantum use on the simulations. The basic quantums mechanism varies from the basic concepts of a quantums (that are used to compare if the new value is on the same region of the previous) up to different models implementations techniques that allow internal quantization mechanisms with important message and time reductions, while incurring into a wide range of errors (depending on the model, the quantum technique and the quantum value).
There are quantum techniques implementing dynamic concepts that dynamically adjust the quantum value of the simulation, depending on the cell values of the simulation, with the possibility of enhance the message reduction and reduce the incurred error. On this case, as the quantum value, a variation rate (which dynamically modifies the initial quantum value) can be also specified for the tool.
During this work, new quantum techniques were implemented and analyzed, like the Hysteresis Quantum [20] were the quantum also varies depending on the direction changes of the values, in order to penalize the behavior of oscillating values changes.
Different implementations of the models are presented and analyzed allowing simulation time and message reduction with approximation techniques (apart of quantization techniques), which are also compared to the quantized models.

We also analyzed GDEVS [14] (Generalized Discrete Event Simulation of Dynamic Systems), which is an approximation technique that keeps the complexity of the rules defined in each cell to a minimum expression. Using GDEVS permitted us to improve the model precision while incurring in fewer time steps when compared with the other quantum methods. The use of GDEVS will also improve the precision obtained if we compare the results obtained by traditional cellular automaton, due to the improved precision of model states. We were also able to obtain models that are very simple in terms of representation. Explicit timing delay constructions permitted us to define precise timing in each cell, which is defined by a local computing function combined with a delay construction.

Another quantum technique was also implemented and analyzed. It's the Adaptive Quantization (named Q-DEVS on this work). Adaptive Quantization [13] is a technique that allow us to predict the next transition for a given rule by computing the next time a cell will achieve the given quantum value. Q-DEVS is a Cell-DEVS model that sets the delay and next value of a cell as a result of the anticipated calculation of the next quantum time. The next quantum time is when a region change will occur. This is calculated using the inverse function of the local updating function of the original model.

The analyzed models with the different quantum types, techniques and values are:

- Heart Tissue [12]: Simulates part of the Heart electrical activity
- Watershed [16]: Simulates rain behavior on different zones (lands)
- Flow Injection Analysis (FIA) [17]: Simulates automated sample analysis of liquid samples

Different versions of these models were implemented and tested with the different quantum types and techniques. The results were analyzed considering the number of messages generated by the tool, the incurred error due to the quantification and the processing time.

The implemented models were made using C++ functions added to the CD++ tool in order to simplify the model design. The implementation of these functions are new techniques (in CD++) to simplify complex implementations and work with n-arguments calculations.
Some revisions were made on the CD++ tool to work properly with the different quantum techniques. E.g. a new argument was added to avoid the use of a 3rd plane. A 3rd plane was previously used to trigger time-based actions for the first plane.

To make the simulations runs and the results analysis, several tools were created. These tools are command line tools to calculate the error (relative and absolute) of a quantized simulation, to get the number of generated messages (by messages type) of a simulation, to see the simulation results on a graphical way and automatically (and massively) run and compare n-simulations.

This work present and analyze the mentioned quantum techniques with the three complex models introduced before.

2 Background

2.1 The DEVS formalism

Systems whose variables are discrete and the time advance is continuous are known as DEDS – Discrete Events Dynamic Systems, as opposed to CVDS – Continuous Variable Dynamic Systems [2].
A simulation mechanism for DEDS systems assumes that the system will only change its state at discrete time points upon the occurrence of an event. An event is formally defined as a change of state that takes place at specific point of time $i_t \in R$.

DEVS [1] is a formalism for modeling and simulation of DEDS systems. It defines a way of specifying systems whose states change upon the reception of an input event or the expiration of a defined time delay. It also allows hierarchical decomposition of the model by defining a way to couple existing DEVS models.

The original DEVS model is a structure:

$$DEVS = < X , Y , S, \delta_{ext} , \delta_{int}, \lambda, ta>$$

Where

X	is the set of external events
Y	is the set of output events
S	is the set of sequential states;
$\delta_{ext}: Q \times X \rightarrow S$	is the external state transition function;
where $Q := \{ (s, e) \mid s \in S , 0 \leq e \leq ta(s) \}$ and e is the elapsed time since the last state transition.	
$\delta_{int}: S \rightarrow S$	is the internal state transition function;
$\lambda : S \rightarrow Y$	is the output function;
$ta : S \rightarrow R_0^+ \cup \infty$	is the time advance function;

The semantics for this definition are as follows: At any point of time, a DEVS model is in a state $s \in S$ and in the absence of external events, it will remain in that state for a period of time as defined by ta(s). The ta(s) function can take any real value between 0 and ∞. A state for which ta(s) = 0 is called a transient state. On the other hand, if ta(s) = ∞, the system will stay in that state forever unless an external event is received. In that case, s is called a passive state. Transitions that occur due to the expiration of ta(s) are called internal transitions. When an internal transition takes place, the system outputs the value λ(s), and changes to state $\delta_{int}(s)$. A state transition can also happen when an external event occurs. In this case, the new state is given by δ_{ext} based on the input value, the current state and the elapsed time.

2.2 Cellular Automata

Cellular Automata are used to describe real systems that can be represented as a cell space. A cellular automaton is an infinite regular n-dimensional lattice whose cells can take one finite value. The states in the lattice are updated according to a local rule in a simultaneous and synchronous way. The cell states change in discrete time steps as dictated by a local transition function using the present cell state and a finite set of nearby cells (called the neighborhood of the cell – Figure 1).

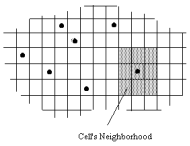


Figure 1 : Sketch of a Cellular Automaton [3]

When cellular automata are used to simulate complex systems, large amounts of compute time are required, and the use of a fixed interval discrete time base add restrictions in the precision of the model. The Timed Cell-DEVS formalism [2] tries to solve these problems by using the DEVS paradigm to define a cell space where each cell is defined as a DEVS atomic model. The goal is to build discrete event cell spaces, improving their definition by making the timing specification more expressive.

2.3 The Timed Cell-DEVS formalism

Cell-DEVS defines cells as DEVS atomic models. A Cell-DEVS atomic model is defined by [2]:

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

where

X	is a set of external input events;
Y	is a set of external output events;
I	represent the model's modular interface;
S	is the set of sequential states for the cell;
θ	is the cell state definition;
N	is the set of states for the input events;
d	is the delay for the cell;
δ_{int}	is the internal transition function;
δ_{ext}	is the external transition function;
τ	is the local computation function;
λ	is the output function; and
D	is the state's duration function.

A cell uses a set of input values N to compute its future state, which is obtained by applying the local computation function τ. A delay function is associated with each cell, deferring the output of the new state to the neighbor cells. There are two types of delays: inertial and transport delays. When a transport delayed is used, the future value will be added to a queue sorted by output time. Therefore, all previous values that were scheduled for output but that have not yet been sent, will be kept. On the contrary, inertial delays use a preemptive policy: any previous scheduled output value, unless the same as the new computed one, will be deleted and the new one will be scheduled. This activation of the local computation is carried by the $\square_{ext}$ function.

After the basic behavior for a cell is defined, the complete cell space will be constructed by building a coupled Cell-DEVS model:

$GCC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z, select >$

where

Xlist	is the input coupling list;
Ylist	is the output coupling list;
I	represents the definition of the interface for the modular model;
X	is the set of external input events;
Y	is the set of external output events;
n	is the dimension of the cell space;
$\{t_1,...,t_n\}$	is the number of cells in each of the dimensions;
N	is the neighborhood set;
C	is the cell space;

B	is the set of border cells;
Z	is the translation function; and
select	is the tie-breaking function for simultaneous events.

This specification defines a coupled model composed of an array of atomic cells. Each cell is connected to the cells defined in the neighborhood, but as the cell space is finite, either the borders are provided with a different neighborhood than the rest of the space, or they are "wrapped", meaning that cells in one border are connected with those in the opposite one. Finally, the Z function defines the internal and external coupling of cells in the model. This function translates the outputs of m^{th} output port in cell C_{ij} into values for the m^{th} input port of cell C_{kl}. Each output port will correspond to one neighbor and each input port will be associated with one cell in the inverse neighborhood.

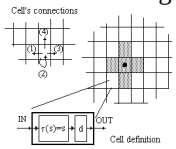


Figure 2 : Informal definition of a Cell-DEVS model [2]

2.4 Cell-DEVS Quantization Theory

A theory of quantized models was developed [4]. When using a quantized model, after a cell's state value will be only informed to its neighbors if its difference with the previous value is greater than a given quantum. This idea is shown in Figure 3. Here, a continuous curve is represented by the crossings of an equal spaced set of boundaries, separated by the quantum size. A quantizer checks for boundary crossings whenever a change in a model takes place. Only when such a crossing occurs, a new value is sent to the receiver. This operation reduces substantially the frequency of message updates, while potentially incurring into error.

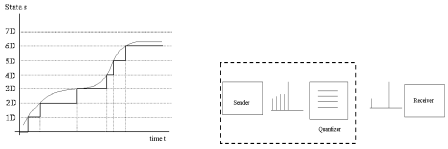


Figure 3 : Quantization (Zeigler et al 1999)

Large experimental tests were done in order to analyze the behavior of quantized Cell-DEVS models. The results showed that quantization reduced both, the total number of messages sent and the execution time, but introduced an error. The error obtained is a function of the local computing function, the number of simulation steps and the quantum. Since the future input values for a cell depend on the present results, a nonlinear error may be observed. The error magnitude will depend on the cell's neighborhood size. It was shown that as the quantum gets higher, the error gets bigger.
Choosing an adequate quantum will then depend on the precision desired.

When quantization is used with a quantum value **d**, δ_{ext} is defined as:

δ_{ext}:

$N_c = \tau_{con}(X^b)$; $(s', transport) = \tau(N_c)$; $\sigma \neq 0$; $e = D(\theta \times N \times d)$; $phase = active$;

$s \neq value(s',d) \Rightarrow$
$(s = s' \wedge \forall i \in [1,m]\ a_i \notin \sigma queue,\ a_i.\sigma = a_i.\sigma - e \wedge \sigma = \sigma - e;\ add(\sigma queue, <s', d>) \wedge f = s)$

$N_c = \tau_{con}(X^b)$; $(s', transport) = \tau(N_c)$; $\sigma \neq 0$; $e = D(\theta \times N \times d)$; $phase = passive$;

$s \neq value(s',d) \Rightarrow (s = s' \wedge \sigma = d \wedge phase = active \wedge add(\sigma queue, <s', d>) \wedge f = s)$

$N_c = \tau_{con}(X^b)$; $(s', inertial) = \tau(N_c)$; $\sigma \neq 0$; $e = D(\theta \times N \times d)$; $phase = passive$;

$s \neq value(s',d) \Rightarrow (s = s' \wedge phase = active \wedge \sigma = d \wedge f = s)$

$N_c = \tau_{con}(X^b);\ (s', inertial) = \tau(N_c);\ \sigma \neq 0;\quad e = D(\theta \times N \times d);\quad phase = active;$

$s \neq value(s',d) \Rightarrow s = s' \wedge (f \neq s' \Rightarrow \sigma queue = \{\varnothing\} \wedge \sigma = d \wedge f = s)$

where

value(v,d) = v' such that $\exists\ q \in N / v' = q.d \wedge v' \leq v$.

i.e. the lowest boundary as defined by the quantum size.

e.g.: value(23.45, 0.1) = 23.4 value(550, 100) = 500

2.5 Quantum hysteresis description

[20] Quantum hysteresis implements a variation that changes the quantum value to the double when there are direction changes on the values.

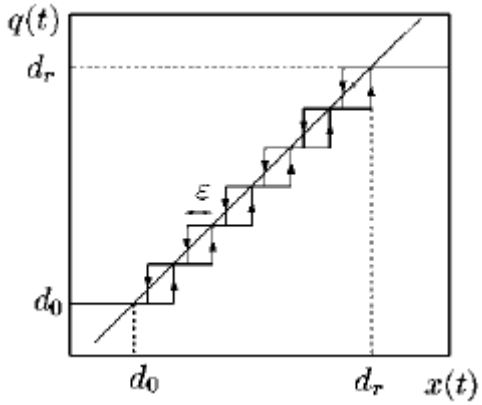


Figure H1 - Quantization Function with hysteresis

Figure H1 shows the quantization function with hysteresis, where:
Let $D = \{ d_0,..,d_m\}$ be a set of real numbers where $d_{i-1} < d_i$
$x \in \Omega$ is a continuous trajectory where:
$x:R \rightarrow R$ and $b: \Omega x R \rightarrow \Omega$ is a maping where $q=b(x,t_0)$ that satisfies:

$$q(t) \begin{cases} d_m & \text{if } t = t_0 \\ d_{i+1} & \text{if } x(t) = d_{i+1} \wedge q(t^-) = d_i \wedge i < r \\ d_{i-1} & \text{if } x(t) = d_{i-E} \wedge q(t^-) = d_i \wedge i > r \\ q(t^-) & \text{otherwise} \end{cases}$$

and

$$m \begin{cases} 0 & \text{if } x(t_0) < d_0 \\ r & \text{if } x(t_0) > d_r \\ j & \text{if } d_j <= x(t_0) < d_{j+1} \end{cases}$$

The map b is the quantization function with Hysteresis. The hysteresis width is E and the parameters d0 and dr and the lower and upper saturation values, as shown in Figure H1.
On this work the hysteresis width was set equal to the quantum size, because of conclusions made in the analysis of the different possibilities [20].

This means that if a value change his direction with respect to the Last Threshold Value, the next value will have to change two regions to be transmitted. This quantum must be combined with standard quantum and can be combined with dynamic quantum too.

When using quantum hysteresis, a value is transmitted depending of this rule:

Let:
q = specified quantum
d = q(t) is the quantum value used in time t
q(0) = q
v = Last Threshold Value
v' = new computed value of the local transition
$\beta(t)$ = direction at time t

A value is transmitted / informed to the neighbors if:

$$\beta(t) \neq \beta(t\text{-}1) \wedge \text{regionChange}(v, v', q*2) \mid \beta(t) = \beta(t\text{-}1) \wedge \text{regionChange}(v, v', q)$$

where

$$\text{regionChange}(a, b, q) = \{ a = \phi \mid q = 0 \mid q \neq 0 \wedge [a / q] \neq [b / q] \}$$

and the direction at time t is:

$$\beta(t) = \{ \text{sign}(v - v') \}$$

the direction at time t-1 is:

$$\beta(t\text{-}1) = \{ \text{sign}(v(t\text{-}1) - v'(t\text{-}1)) \}$$

2.6 Adaptive Quantization description

Adaptive Quantization [13] is a technique that uses the inverse function of the local computing function to get the time when the function will change the region.
For a given function x' = f(x, neighborhood(x), t), if we find f^{-1} / t = f^{-1} (x, neighborhood(x), q), we can use this technique to hardly reduce the number of messages of a simulation designing the model to work setting the delay and the next value with the results of the f^{-1} function.

Q-DEVS is a Cell-DEVS model that sets the delay and next value of a cell as a result of the anticipated calculation of the time when a region change will occur using the inverse function of the local updating function of the original model.

A typical Q-DEVS rule will be:

{ INV(Q, T, x,y,z,….) } { DELAYINV(Q, T, x,y,z,…) } { t }

Where:

INV(Q, T, x,y,z,…) returns the value that the analyzed cell will have on the next region change with respect to the quantum value Q after time T.

DELAYINV(Q, T, x,y,z,…) returns the necessary delay for the analyzed cell to have the value that changes its region after time T with respect to the quantum value Q.

These are theoretical functions. For the analyzed models on this work, the functions where implemented using a different technique to return both values (next value and next delay) with only one inverse function.

2.7 GDEVS description

[14] The definition of cellular models are presented, in which we are able to define individual cells using Partial Differential Equations (PDEs). We use Cell-DEVS to create cell specifications in which cell runs a small portion of a complex system of PDEs. The researchers will be able to focus in defining smaller portions of a problem and in expressing it using simpler differential equations, which can be solved easier than the complete system. We use GDEVS (Generalized Discrete Event Simulation of Dynamic Systems) to keep the complexity of the rules defined in each cell to a minimum expression. Using GDEVS permitted us to highly improve the model precision while incurring in fewer time steps when compared with traditional numerical methods. The use of GDEVS will also improve the precision obtained if we

compare the results obtained by traditional CA, due to the improved precision of model states. We were also able to obtain models that are very simple in terms of representation. Explicit timing delay constructions permitted us to define precise timing in each cell, which is defined by a local computing function combined with a delay construction.

GDEVS is a formalism for the specification of discrete event models of dynamic systems. The originality of GDEVS stems from the use of polynomials of arbitrary degree (as opposed to constant values), to represent the piecewise input-output trajectories of a discrete event model. In essence, GDEVS constitutes a generalization of the classical discrete event modeling approaches including DEVS, in that a classical model may be viewed as a GDEVS model of order 0 (the trajectories are represented by polynomial of order 0). Classical discrete event abstractions of dynamic systems are based on the mapping of piecewise constant input-output segments of (obtained perhaps through threshold sensors) into discrete events. GDEVS adopted a radically new approach based on a new definition of the concept of event ([18], [19]). In GDEVS, the target real-world system is modeled through piecewise polynomial segments. If we note that the polynomial coefficients have piecewise constant trajectories we can built an discrete event abstraction in the coefficient space using the concept of coefficient event. A coefficient event is thus considered as an instantaneous change of, at least, one of the value of the coefficients defining the piecewise polynomial trajectory of the considered variable. An event is a list of coefficient values defining the polynomial that describes the trajectory of the variable.
We intend to extend the basic behavior provided by Cell-DEVS atomic models to permit the users to specify cells with continuous variable behavior using the GDEVS formalism. GEDVS considers the general case of dynamic systems with piecewise continuous input-output trajectories, and it has solved how to transform these piecewise continuous trajectories into discrete event trajectories. This transformation was done by achieving a partition of the output trajectory into piecewise polynomial segments. To each of these output segments corresponds a continuous segment of the state trajectory and piecewise constant segments in the space of polynomial coefficients. In a GDEVS model an event is an instantaneous change in at least one of the values of the coefficients of the polynomial describing the signal.

For example, let us consider a piecewise linear trajectory w<t0;tn> →A as a trajectory on a continuous time base, characterized as follows: there is a finite set of instants $\{t_0, t_1, \ldots, t_n\}$ associated with constant pairs $(a_i; b_i)$ such that $\forall t \in \langle i_t; t_j \rangle$, $w(t) = a_i\, t + b_i$, and $w\langle t_0; t_n\rangle = w\langle t_0;t_1\rangle * \; w\langle t_1;t_2\rangle * \ldots .. \; w\langle t_{n-1};t_n\rangle$ (where * represents the operator left concatenation of segments). This is exemplified in Figure 30, which describes the use of piecewise linear approximations of the continuous segment in Figure 30a, while Figure 30c represents a 1-order discrete event abstraction under GDEVS with coefficient events.

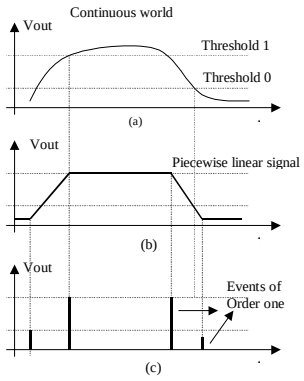


Figure 30. GDEVS approximation of a continuous signal
(a) continuous segment (b) piecewise linear segment (c) 1st order model.

The behavior of each cell in a Cell-DEVS model will now be described using GDEVS. Before applying this new proposal, Cell-DEVS state variables already permitted using continuous values. Nevertheless, continuous functions definition was, in general, constrained to defining discrete time versions of the PDEs (Partial Differential Equations) running in each of the cells. This desvirtuated two of the main advantages of using Cell-DEVS: advancing model execution using discrete events, specifying cellular models as a composite of cells described with very simple rules. In certain cases, we were able to describe continuous functions using ad-hoc simple rules, but most researchers would prefer defining the cell's behavior using PDEs (Partial Differential Equations), which would result in performance degradation. We

will show how now Cell-DEVS models can overcome these problems. The idea is that a continuous local computing function τ will be approximated by piecewise polynomial signals of the desired precision, providing means for improved performance while having simpler rules for model definition (namely, concatenation of polynomials).

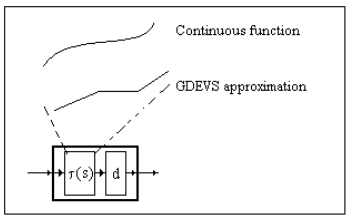


Figure 31. GDEVS approximation of Cell-DEVS local computing functions.

The ideal case in terms of performance is when a linear approximation is able to provide high precision and bounded error, as linear models have low cost of execution and easy definition. Higher precision can be achieved by using higher order polynomials, paying the cost of execution time and increased complexity of the rules defined. Defining GDEVS models using polynomials of order 0 result in automatic definition of traditional CA.

2.8 CD++

CD++ implements the DEVS theory. It allows defining models according to the original DEVS formalism (Wainer 2000-Rodríguez and Wainer 1999). A set of independent applications related with the tool allows the user to have a complete toolkit to be applied in the development of simulation models.

The tool is built as a hierarchy of classes, each of them related with a simulation entity. Atomic models can be programmed and incorporated into a basic class hierarchy programmed in C++. Coupled and Cell-DEVS models need not be programmed. The tool provides a specification language that defines the model's coupling, including the initial values and external events, and the local transition rules for Cell-DEVS models.

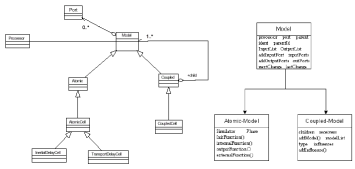


Figure 4 : CD++ Models and Processors.

This class hierarchy implements the model theoretical definition presented in the previous section. New atomic models must be incorporated to the class hierarchy as subclasses of the Atomic Model class. Coupled models are defined using a specialized specification language. Following, we explain how to incorporate atomic and coupled models to be simulated.

A new atomic model is created by including a new class that inherits from Atomic. In doing so, the following methods may be overloaded:

- initFunction: this method is invoked when the simulation starts. It allows to define initial values and to execute any initialization procedure for the model. When this method is executed, the value of sigma next scheduled event) is set to infinite and the model phase to passive. The sigma variable is used to implement the duration function: it stores the time up to the next event in the model. This variable is related with the elapsed time value, which is maintained by an independent simulation mechanism.
- externalFunction: this method is invoked when an external event arrives from an input port. If the new CD++ argument "-c" is indicated, an additional external event is generated when no events are generated on a time for a cell, to keep the model simulating until the ending time.
- internalFunction: this method is started when the value of sigma is zero, since an internal event has occurred.
- outputFunction: this method executes before the internal function, allowing to provide outputs for the model. After defining these functions, new models can be incorporated to the modeling

class hierarchy. Finally, the model must be registered using the method MainSimulator.registerNewAtomics().

The following primitives can be used in defining the atomic's model behavior:

- holdIn(state, time): a model executing this sentence will remain in state during time. When the time is consumed (sigma = 0), the model executes the internal transition. This macro was included to make easy the definition of the duration function.
- passivate(): the model enters in passive mode (phase = passive; sigma = infinite) and it will be reactivated by an external event.
- sendOutput(time, port, value): it sends an output message through the given port.
- state(): it returns the present model phase.

CD++ Quantization implementation

In CD++ there are implemented two main quantum types, which can be combined to obtain different quantification results. The quantum types are Standard quantum and Dynamic quantum. These mechanisms were modified in order to solve some simulation problems with quantization and reduce the error while using a quantum.

The solved problems were three:

The updating of the local value of a cell when the quantum is not achieved.

A way of storing the last value a cell had before changing its region.

The comparison of the values considering the precision indicated on CD++.

The idea of a threshold value was implemented to store the last value a cell had before its last region changed, to be compared with the new value. This value is called Last Threshold Value.

The region is calculated as follows:

v = computed value

d = quantum value

Region:

If $\mathbf{d} \neq 0 \Rightarrow$ Region = [v / **d**]

If $\mathbf{d} = 0 \Rightarrow$ Region = v

Where "[x]" is the integer part (without decimal values) of the x value.

When a value is on the same region of the Last Threshold Value, the new value is not queued (or transmitted) and the neighbors of the computed cell won't receive this information but the local value of the cell it is internally updated for next computation.

With these changes, the Last Threshold Value its now stored and δ_{ext} work as follows:

δ_{ext}:

$N_c = \tau_{con}(X^b)$; (s', transport) = $\tau(N_c)$; $\sigma \neq 0$; $\alpha=\phi$; e = D(θ x N x d); phase = active;

---

regionChange(α, s',d) $\Rightarrow$
(α = s' $\wedge$ α=s' $\wedge$ $\forall$ i $\in$ [1,m] $a_i \notin \sigma$queue, a_i. $\sigma = a_i.\sigma$ - e $\wedge$ $\sigma = \sigma$ - e; add(σqueue, <s', d>) $\wedge$ f = α)

$N_c = \tau_{con}(X^b)$; (s', transport) = $\tau(N_c)$; $\sigma \neq 0$; e = D(θ x N x d); phase = passive;

---

regionChange(α, s',d) $\Rightarrow$ (α = s' $\wedge$ σ = d $\wedge$ phase = active $\wedge$ add(σqueue, <s', d>) $\wedge$ f = α)

$N_c = \tau_{con}(X^b)$; (s', inertial) = $\tau(N_c)$; $\sigma \neq 0$; e = D(θ x N x d); phase = passive;

---

regionChange(α, s',d) $\Rightarrow$ (α = s' $\wedge$ phase = active $\wedge$ σ = d $\wedge$ f = α)

$N_c = \tau_{con}(X^b)$; (s', inertial) = $\tau(N_c)$; $\sigma \neq 0$; e = D(θ x N x d); phase = active;

---

regionChange(α, s',d) $\Rightarrow$ α = s' $\wedge$ (f $\neq$ s' $\Rightarrow$ σqueue = {$\varnothing$} $\wedge$ σ = d $\wedge$ f = α)

$\neg$ regionChange(α, s',d) $\Rightarrow$
($\forall$ i $\in$ [1,m] $a_i \notin \sigma$queue, a_i. σ = a_i.σ - e $\wedge$ σ = σ - e; $\wedge$ f = s')
where
regionChange(a, b, q) = { a = ϕ | q = 0 | q $\neq$ 0 $\wedge$ [a / q] $\neq$ [b / q] }
e.g.:
regionChange (22.0, 23.45, 0.1) = true regionChange (23.49, 23.45, 0.1) = false

Standard CD++ Quantization

The standard quantum works with a quantum value, which is used to compare the new value with respect to the last value that achieved the quantum (called last threshold value) and determine if the new value is on the same quantum region of the last one. If this is the situation (the new value is on the same region of the last threshold value) the new value is not transmitted (informed) to the affected neighbors of the analyzed cell with the message savings that this implies. When a new value is on a different region (with respect to the quantum value and the last threshold value), the new value is transmitted (informed) to the affected neighbors and this value will be the new last threshold value for future region comparisons (on this case, there are no message savings).

Dynamic CD++ Quantization

Dynamic quantum is a variant that changes the current quantum value on a specific ratio. The quantum used at time t can be different of the quantum used at time t´ for the same simulation.
The ratio is applied to the original quantum value. With the ratio, the quantum will change on the range q * (1 – ratio) and q * (1 + ratio).
Dynamic quantum must be combined with standard quantum and with one of its two strategies:
Strategy 1: With this strategy, q * (1 - ratio) is used when the new value is on the same region of the Last Threshold Value and q * (1 + ratio) is used when the new value is on a different region of the Last Threshold Value.
Strategy 2: is the opposite of Strategy 1. It use q * (1 - ratio) when values are on a different region and q * (1 + ratio) when they are on the same region.

Let q be the specified quantum and r the ratio.
d = q(t) is the quantum value used in time t.
v = Last Threshold Value
v' = new computed value of the local transition

q(0) = q for both strategies.

Strategy 1
$\neg$ regionChange(v, v', d) $\Rightarrow$ d = q * (1 - ratio);
regionChange(v, v', d) $\Rightarrow$ d = q * (1 + ratio);
Strategy 2
regionChange(v, v', d) $\Rightarrow$ d = q * (1 - ratio);
$\neg$ regionChange(v, v', d) $\Rightarrow$ d = q * (1 + ratio);

where
regionChange(a, b, q) = (a = ϕ | q = 0 | (q $\neq$ 0 $\wedge$ [a / q] $\neq$ [b / q]))

3 CD++ Modifications

Some changes on CD++ tool were needed to make the quantization analysis made on this work.
The first change was the change to store the Last Threshold Value and to update the current cell value also when no region change is given while using quantum.
Previous to this work, when a quantum value was in use, the error was very high with small quantum values (specially on models using the current cell value to calculate the next value, that is, when the current value is an input of the local computation function). A problem detected was that CD++ was not

updating the current cell value when the new value didn't changed its region with respect to the last value. When this occurred, the error was to big because it increased the difference with the original value because the local computation function used the previous value to calculate the new cell value and that previous value was not updated since the last region change.
A possible solution was to store internally the last calculated value for the analyzed cell, but this won't solve all the problems with all the models, so a better solution was implemented: When a value didn't pass the region of the last value, the current value of a cell is updated locally, without sending messages to its neighbors. At this way, in spite of not queuing and sending the new value who didn't passed the quantum, the current cell updated its value internally and no new events were generated or scheduled. Because of this change, a new consideration to keep in mind, was that the value of the cell is now always updated (passing or not the quantum) so it's now necessary to know which was the last value passed a quantum or changed the region (because the current cell value is not, it's updated always) and that value (the last passed a quantum / changed the region) is the value to compare with the new value with the quantum. This was the reason because the Last Threshold Value was added to the tool and after this change, the incurred error while quantifying was reduced significantly and the quantum analysis could take place.
Another change made to the tool was the quantum comparison function. When quantum was in use, some values which it suppose that will pass the quantum won't. The reason was that the comparison function was not using the "tolerance" argument of the tool and this implied that comparisons fails more than the expected because of precision problems. A new comparison function was implemented which uses the simulation specified precision (tolerance) on the tool to compare the values. With this implementation, the precision for quantum region comparisons can be managed with the tool arguments.
After this changes, when using dynamic quantum, the error and the results were very high comparing the dynamic quantum with non dynamic quantum. A new modification was needed on CD++ in order to solve this problem. The dynamic quantum was updating the quantum value with the specified ratio value on the last updated quantum, not on the original quantum. A modification to use the original quantum to apply the ration value to update the quantum was introduced and dynamic quantum could be analyzed correctly.
Another issue when using dynamic quantum was when quantifying one plane, the dynamic quantum value was updated also with the non quantified plane. To solve this, updating logic of dynamic quantum was modified so when other plane than the selected for quantifying is evaluated, the dynamic quantum does not update the quantum value in order to don't disturb the quantum with this plane who was not selected for quantification. Similar problems occurred with quantum Hysteresis combined with dynamic quantum: the hysteresis quantification considers different quantum conditions depending on the direction changes of the values. The dynamic quantum updating had to follow the same rules: if a value does not achieve the double quantum because direction changes took place, the dynamic quantum updating rule has to be done in accordance with the dynamic standard quantum when it doesn't achieved the quantum (in spite of a double quantum). With this changes, hysteresis with dynamic quantums were synchronized in order to work properly.
Some other changes in order to enhance the showed and validation of the arguments during a simulation were made. Selected quantum information and the modifiers of quantums (like the selected plane for quantifying) were added. Also quantum validations were added, e.g., if dynamic quantum is needed, type, value and ratio has to be specified, because when using dynamic quantum is not enough to indicate the arguments of dynamic quantum, also the quantum type and value are needed to be combined (e.g. Standard or Hysteresis values).

New Quantum hysteresis Argument

A new argument was added to support this technique. The argument is –Q for quantum hysteresis while –q is used for standard quantum. These arguments can be combined with the arguments of dynamic quantum, which are: -y for strategy 1 and –Y for strategy 2.
e.g.:

A valid argument usage can be:

-Q1.5 –y0.5

which means using quantum hysteresis 1.5 and dynamic quantum strategy 1 with dynamic ratio 50%.

With this argument, when the direction changes (direction is the sign of {newValue-oldValue}), the newValue will have to achieve the double quantum. This behavior will occur when the new argument "Q" (similar to "q" of standard quantum) is indicated. To support this mechanism, CD++ changes were implemented in some simulator classes.

This is the behavior of quantum hysteresis:

1. If lastThresholdValue is undefined, it means that is the first comparison and lastThresholdValue has to be initialized with the current value of the cell and the direction of the cell will be the sign of the difference with the initial value of the cell and this new value (0 at the beginning).
2. Comparing the current (stored) direction of the cell values with the new direction, if the direction is the same and the absolute value of the difference between the lastThresholdValue and this new value with respect to the quantum is different, lastThresholdValue is updated with this new value, the new direction between this value and the current value is updated and the value is scheduled (queued in case of transport delay).
3. If the direction is the same and the absolute value of the difference between the lastThresholdValue and this new value with respect to the quantum is equal, lastThresholdValue and the direction are not updated and no value is scheduled (or queued) but the local cell value is internally updated with the new value ins spite of not achieved the quantum.
4. If the direction is different and the absolute value of the difference between the lastThresholdValue and this new value with respect to the double of the quantum (2*quantum) is equal, lastThresholdValue and the direction are not updated, neither a new value is scheduled but the local cell value is internally updated with the new value in spite of not achieved the double required quantum.
5. If the direction is different and the absolute value of the difference between the lastThresholdValue and this new value with respect to the double of the quantum (2*quantum) is different, lastThresholdValue and the direction are updated and the new value is scheduled (or queued in case of transport delay cell).

The steps showed above are a description of the hysteresis implementation in CD++.

New Quantum slide selection Argument

A problem solved here was the selection of the slide to quantify.
On a Cell-DEVS model, it is possible to have 3-dimensions or n-dimensions for a model.
e.g.:

to quantify the original model of Watershed, a 3rd plane was necessary to maintain the cells with activity, because if no queued values were available for a cell, the cell will be inactivated and if no events are available for none of the cells, the simulation will stop as a result of no events generation.

To bypass this problem and analyze the quantization, a 3rd plane was added to keep the model with events. The next rules are an example of the mentioned 3rd plane rules:

Rule : { 1 } d { (0,0) = 0 and Plane = 2 }
Rule : { 0 } d { (0,0) = 1 and Plane = 2 }

With these, we saw that for some models, could be necessary to quantify only some planes (or some cells) of the model and not every cell and every plane, so with a new argument, we can now specify which cells or planes to quantify, as follows:

-i(X:n*,X:n*,….X:n*)

where –i indicates CD++ that not every cell or plane has to be quantified, $X \in$ ("S", "N"), $n \in N_o$
X = "S" indicates that this plane must be quantified.
X = "N" indicate that this plane mustn't be quantified.
n indicates that only the nth plane has to be quantified or not quantified (it depends on the X value, if it's S means quantify, N means not quantify) and the enumeration of n is optional to indicate specific dimensions cells.
e.g.:

-i(S,S,N:1) means that every cell will be quantified except cells on plane 0 of last dimension. Cells of form (*,*,1) wont be quantified. Any other yes.

-i(S,S,S:0) means that every cell of form (*,*,0) will be quantified. Any other no.

-i(S,S,S:0,2) means that every cell of form (*,*,0) or (*,*,2) will be quantified. Any other no.

This is useful to quantify models containing rules for cells or planes that we don't want to quantify.

New Discrete advance time simulation Argument

With a new argument (-c), we can indicate CD++ that this simulation will force the continuity of simulations also when no more events are given if the ending time was not achieved. The simulation will end only when simulation time achieves the ending time indicated for the simulation.
This argument receives also the interval of time used to discretize the simulation when no events are given for a cell.
The format of the argument is –cε where ε is the time interval.

e.g.:
-c00:00:00:200 means that the simulation will continue on a cell with no events every 200ms.

When no events are generated at a time t for a cell, CD++ will generate an external event in t+ε time.

Drawer (drawlog) changes

As in the simulation, drawlog tool (used to get an output file with the values of each cell from the log of the simulation) uses an argument to extract only the output corresponding to an specific plane.
Drawlog already has an argument –f to indicate this, but this argument keeps out all the format information (titles and timeline) and these argument outputs values on every time the simulation generates an output, without filtering if the output is for the selected plane or not. This was useful for the graflog tool, which needs no lines and time formats to work.

With the "–e" new argument drawlog now can generate a formatted output for the selected plane and only when the selected plane has values, ignoring the times outputs of others planes and keeping on the titles, time and cell numbers headings.
This is useful for a new graphic tool that shows a graphic of the local function of each cell on a cell space grid indicating the time and cell on the graphic.
Also to get the error comparison of simulations, the time is needed to compare correctly the values of each time.
In drawlog, an "-i" argument was added which indicates the number of milliseconds to show the output. This is used to analyze large simulations, with very big output files, because it allow "jumping" messages in between.

4 Heart Model Description and Implementations

Description
[7] The heart is a muscle responsible for pumping of blood into the circulatory system. Behavior of the phenomena occurring in the heart muscle and tissue has been extensively studied and it has been reported in a wide variety of medical treaties (see, for instance [5], [6]). In these documents, heart activity is usually analyzed according to three kinds of activities: mechanical, electrical and cellular.

In terms of mechanical activities, the blood returns to the heart through the vena cava superior and inferior, and flows to the right atria. The blood flows to the right ventricle, where it is pumped to the lungs to return oxygenated to the left atria. Then, it flows to the left ventricle, which returns the oxygenated blood to the body through the aorta. This is presented in the following Figure 5.

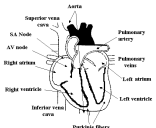


Figure 5. Basic anatomy of the heart.

Mechanical activity is triggered by the electrical activity of the cells. The heart muscle is excitable, and the cells in its tissue respond to external stimuli by contracting the muscular cells. If the stimulus is too weak, the muscle does not respond; instead, if the voltage received is adequate, they contract at maximum

capacity. The electrical conduction system of the heart is responsible for the control of its regular pumping. This activity is originated in the SinoAtrial (SA) node, also known as the pacemaker. This is an electrically active region of the heart that self activates automatically. Cells in the heart tissue are excited when adjacent cells are charged positively. In that case, an upstroke of its action potential is provoked, which will spread to nearby cells.
All excitable tissue, once activated, exhibits a refractory period before returning to rest. During the contraction period, the muscle is absolutely refractory, and do not respond to external stimuli. Before starting a new contraction, the previous one should have finished, and shortly after the contraction, the muscle is relatively refractory. In this case, minimum stimuli do not generate response, but a stronger stimulus is able to generate a response. The electrical activity is started in the SA node, and it spreads through the atria muscle at a speed of 1 m/s (for human beings 80 ms are needed to activate the atria). After, the electrical activity is spread to the AV node, where it propagates slowly (0.1 m/s), and then, the excitation travels at 2 m/s through the Purkinje fiber.

This electrical activity is originated by the cellular activities, which consists on the interchange of ions of Potassium and Sodium in the walls of the cells. This chemical reaction produces potential differences of mV, which trigger the electrical activity. Hodgkin and Huxley originally characterized this behavior of cell membrane activity in [8], a foundational article that presented the detailed behavior of the inter-membrane action potential function. They recognized different phases in this function:

a. The heart tissue is relaxed, the interior of the membrane is electrically negative with relation to the surface, with a difference of potential of 50 mV
b. The surface membrane is repolarized, creating two zones with a potential difference.
c. Electrical activity starts, and the external surface became negative, with a potential difference of 30 mV. This phase is called excitation (or depolarization).
d. Finally, negative voltage in the surface trespasses the membrane, and the original status is recovered. This phase is called repolarization.

The Hodgkin-Huxley model showed that virtually all membrane current models can be defined by writing the total membrane current, which is a sum of the individual currents carried by different ions through specific channels in the cell's membrane. The calculation is based on Sodium ion flow, Potassium ion flow, and the leakage ion flow. This behavior can be defined as:

$$I = m^3 h G_{Na} (E - E_{Na}) + n^4 G_K (E - E_K) + G_L (E - E_L) \quad (1)$$

I is the total ionic current across the membrane;
m is the probability that one particle contributed to activate of the Sodium gate;
h is the probability that one inactivation particle has not caused the Sodium gate to close;
G_{Na} is the maximum Sodium conductance;
E is the total membrane potential;
E_{Na} is the Sodium membrane potential;
n is the probability that 1 of 4 particles influenced the Potassium gate;
G_K is the maximum possible Potassium conductance;
E_K is the Potassium membrane potential;
G_L is the maximum Leakage conductance; and
E_L is the leakage membrane potential.
Hodgkin and Huxley computed empirical formulas for the Sodium gate activation (m), Sodium particle activation probability (h), and Potassium gate activation probability (n). They also found the values of the remaining parameters of equation (1), which where shown to be constant. By applying the Hodgkin-Huxley equations, we can obtain the action potential function for the cells in different regions of the heart tissue. The behavior of different cells can be defined by variation in conductivity, length of the fibers, etc. They also showed that the results of this equation are equivalent to the results found in experimental data. The following Figure 6 shows the results obtained when using the Hodgkin-Huxley equations using parameters corresponding to cells of the atria. We will use this equations in following sections to build the Cell-DEVS model of the hear tissue, and to make the quantization analysis.

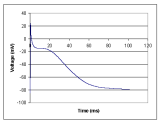


Figure 6. Action potential in the atria cells using Hodgkin-Huxley equations.

CD++ Implementations

The Hodgkin-Huxley model has been extensively used in different studies, as it has been shown that it reproduces with fidelity the electrical properties in the myocardium cells. Nevertheless, whereas solving this equation using numerical methods for one cell is feasible, the use of this model in a realistic reproduction of the heart tissue (probably consisting of millions of cells) can be computationally expensive. Consequently, different authors tried to simplify the complexity of the equations, and various studies tried to solve this problem using CA (see, for instance, [9], [10], [11]). Most of these models are based on simple CA for excitable media, which discretize the Hodgkin-Huxley results.

In [10], we used Cell-DEVS to build a discrete variable model of heart tissue conduction. In this model (which uses a similar approach than other models built using CA), we recognized three states for a cell: resting, excited or recovering. We defined this model in the CD++ toolkit, and Figure 7 includes a complete specification of it.

```
[Heart]
type : cell
dim : (5,5)
delay : transport
border : nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)
neighbors : (0,1) (1,-1) (1,0) (1,1) (0,0)
localtransition : Heart-rules

[Heart-rules]
rule : 2              0.48    {(0,0)=0 and statecount(2)>0 }
rule : 1              1.48    { (0,0) = 2 }
rule : 0              17.5    { (0,0) = 1 }
rule : { (0,0) }      0       { t }
```

Figure 7. Cell-DEVS Definition of a simple heart tissue model.

Model definition begins by defining the Cell-DEVS coupled model and its parameters: size (5x5 cells), neighborhood shape (all of the adjacent cells), kind of delay (transport, as we want every state change to be transmitted without preemption), and borders (this is a non-wrapped model, and special rules were defined for the borders). The heart-rules section represents the local computing function for the model. Here, the first rule represents the initiation of electrical activity in a resting cell (with value 0). In that case, we check to see if any of the neighbors is excited (value 2). In that case, the cell is excited. Second and third rules define the cells changing to the recovering and resting states. The last rule states that in every other case (t means "True"), the cell keeps its present state. Figure 8 shows the results obtained when this model executes. It shows the evolution of this considering a pacemaker cell in (0,0).

Figure 8. Heart tissue model execution.

As we can see, the model represents the tissue action using very simple rules, which has several advantages. Cell behavior is defined using simple rules, which makes it easy to modify the existing model to experiment different conditions. We also see that delay functions are associated to each of the rules representing each state cell. When describing this model using CA, timing definition is more complex, and it can result in extensive simulation time to achieve the desired precision. Likewise, any changes in the delay functions can result in complex changes in the CA definitions. Instead, Cell-DEVS timing delays can provide complex timing description using rules that are straightforward to define. For instance, this model represents three different delays at different scales. In order to achieve such precision in CA we should choose the smallest timeslot for simulating time advance. Instead, in Cell-DEVS each rule is triggered by an event that is executed asynchronously in each of the cells at randomly chosen instants.

Although representing this problem as CA permits introducing simple rules, it poses a problem in model's precision. As we can see, we have discretized the continuous function showed in Figure 6 with only three different discrete states. This could seriously affect a the execution results of the model if we needed to introduce modifications to the cell's standard behavior. For instance, arrhythmias modify the action potential in isolated groups of cells, modifying the shape of the action potential curve, which could require defining a completely erratic behavior for a group of cells, which could affect the adjacent cells under different circumstances. Another example considers analysis of the cell's behavior during the

refractory period: if enough voltage is received on a cell, the cell is excited, but if the voltage is not enough, it will ignore the stimuli. Representing this behavior with CA is very difficult; instead, if a PDE (Partial Differential Equations) is included on each cell, it will be able to adequately react to each possible modification of the parameters.

As a result, we decided to implement this model as Cell-DEVS running the Hodgkin-Huxley model in each of the cells. We implemented a model of the Action Potential function for the cells in the heart atria [12]. This Cell-DEVS model simulates the electrical behavior of the cells, following the Hodgkin-Huxley model, as described in section 3, discretizing the time in each of the cells under execution. The following Figure shows the model definition using CD++.

```
[heart]
type : cell
dim : (5,5,2)
delay : transport
border : nowrapped
neighbors : (-1,-1,0) (-1,0,0) (-1,1,0)
neighbors : (0,-1,0)  (0,0,0)  (0,1,0)
neighbors : (1,-1,0)  (1,0,0)  (1,1,0)  (0,0,1)
localtransition : heart-rule-AP

[heart-rule-AP]
rule : { AP(cellpos(0) } 1 { cellpos(2)=0 and (
                    (-1,0,0) > 0 or (0,-1,0) > 0 or (-1,-1,0)>0) and (0,0,0) = -83.0) }
rule : { AP(cellpos(0) } 1 { cellpos(2)=0 }
rule : { if( (0,0,0) = 1.0 or (0,0,0) = -83.0, 0.0, 1.0) } 1 { cellpos(2)=1 }
```

Figure 9.Cell-DEVS definition of the action potential function for a heart tissue model [12].

We first define the size of the cell space, which, in this case, a 3D model with 5x5x2 cells. This model uses a transport delay, and it is non-wrapped (we define specialized behavior for the cells in the border). The following lines in the specification define the neighborhood shape (in this case, all the adjacent cells in plane 0, and the upper cell, which will be used to define if the current cell should be computed or not). Then, we define the local computing function, called heart-rule-AP. The function is defined by two rules. The first one will be evaluated only by the cells in the first plane in the model (cellpos(2)=0), and only if the cell is resting and a positive voltage is detected in the cell's neighborhood. This rule will trigger the update of the cell state using the Hodgkin-Huxley equations presented in Section 3. The second rule will be used in the subsequent activations. The third rule is evaluated only by second plane (cellpos(2)=1), and it is used to trigger time-based actions for the first plane. This plane is just changing its state from 0 to 1 and vice versa in each timestamp, triggering the execution of the rules of the action potential function in plane 0. This is needed because Cell-DEVS only considers activation of a cell under asynchronous events, and, if no event is created, the cell goes to a quiescent state, which is avoided by this rule.

(a) (b)

Figure 10. Model execution using Hodgkin-Huxley equations (a) individual cell (b) cell space

The AP function in this model receives the coordinates of the current cell and its current state. Using these values, it recovers the previous state of the current cell, and computes the next voltage using equation (1). Figure 10 shows the execution results of this model. As we can see, the results obtained are the same we obtained earlier by solving analytically the Hodgkin-Huxley (in fact, most of the source code originally developed to build the AP function was reused in this Cell-DEVS model).

As we can see, this model improves precision over the CA, thus, we are able to define advance cell behavior easily. For instance, by activating the AP function with different parameters in different cells, we are able to reproduce the activity in sick cells (like those with arrhythmias, fibrillation or conductivity problems). Nevertheless, this model is very expensive in terms of computing resources.

To analyze the different techniques of quantum (and mechanisms), three quantum variants where implemented:

The standard quantum (as showed before).
The Q-DEVS quantum (for Adaptive Quantization).
The GDEVS models.

Here are the details of those implementations:

New CD++ Implementation

The implementations were made using a function written in C++ inside the source code of CD++ tool (the AP function).

This code makes easier the implementation of this kind of models within complex functions. The behavior of the neighborhood is described on the CD++ model using the implemented function in it. To analyze correctly the model with the different quantum mechanisms, the model was simplified reducing the neighborhood rules to the minimum (one rule) and avoiding the neighbor restrictions present on the original model for cell activation (the restriction was the comparison of some neighbors values with zero value, to verify if the neighbors has positive or negative values). With this simplification it's possible to compare the simulations correctly, with the different quantum types and techniques.

Above we can see the model implementation, which has to be run with a new CD++ argument which indicates the tool that discrete time is in use at intervals of x ms (5 ms in this case). With this argument, if on a period no events are generated for a cell, the simulation will go on with an external transition generated by an external message to the cells, sent by the cell with no events itself. This is useful to avoid having the 3rd plane used before, which disturbs the message comparison and simulation time on the analysis.

Implementing this changes, we have a 2nd model with less complexity of rules:

```
%modelo del Corazon
%-----------------------------------------------------------------
%Ejecutar con parametros -c00:00:00:005 (discrete time tik 5ms)
%-----------------------------------------------------------------

1   [top]
2   components : corazon

3   [corazon]
4   type : cell
5   dim : (5,5)
6   delay : transport
7   defaultDelayTime  : 5
8   border : nowrapped
9   neighbors :  corazon(-1,-1) corazon(-1,0) corazon(-1,1)
10  neighbors :  corazon(0,-1)  corazon(0,0)  corazon(0,1)
11  neighbors :  corazon(1,-1)  corazon(1,0)  corazon(1,1)
12  initialvalue : -85.0
13  initialCellsValue : corazon.val
14  localtransition : corazon-rule

15  [corazon-rule]

%Como parámetro recibe el voltaje actual de la celda y un indicador de si alguna de las
celdas vecinas
%tiene corriente positiva cuando esta celda esta en reposo.
%El ultimo valor sumado es la demora utilizada.

16  rule : { APA(cellpos(0)*10000000+cellpos(1)*100000+0*10000+0.025,(0,0)) } 5 { t }
```

On **Lines 1 and 2** the unique Cell-DEVS model corazon is defined.

On **line 3** the model description begins.

Line 4 indicates CD++ that is a cellular model.

Line 5 is the defined dimension of the cellular space. In this case, we have a space of 5x5x1 cells (on previous models, a 3 dimensional space of 5x5x2 was needed to keep alive the model wile quantifying).

L**ine 6** defines the delay type of the model for each cell (tissue). In this case, the delay type used is transport.

Line 7 defines default delay, 5 milliseconds in this model and it's the same delay for every cell (this is not necessary true on the real model, but is sufficient to make the simulation analysis needed for quantum).

Line 8 defines the border type for the model, which is no wrapped, which means that the border cells are not connected with the opposite border (as is on a heart tissue space).
Lines 9 to 11 defines the relative positions of the neighbors which are all the adjacent cells including the analyzed cell.
Line 12 defines the default value for every cell, which is –85.0 for all the cells because we are analyzing the model as a uniform space, which is necessary to compare with other quantum types.
Line 13 indicates the file with the initial values for each cell, which in this case is the same (–85.0) but in other models tested at the beginning was different to indicate the pacemaker cell, as in the heart, which has a pacemaker fiber which initiates the action potential of the heart, but now, we are not using a pacemaker in order to keep a uniform simulation for all the cells.
Line 14 is the name of the section with the behavior rules for the cellular model.
Line 15 is beginning of the rules description section.
Line 16 is the first rule (and the unique in this case). This rule defines the local updating function of each cell, which represents each tissue of the heart.
On previous models, an additional rule was used to keep alive the model wile using quantum, but here is no more needed.

The APA function used on line 16 has 5 arguments. The first four arguments are composed because of a limitation of the used version of CD++.

On this implementation, the initial values for all the cells are –85.0 in order to have the same behavior on all the cells and avoid comparison problems (complexity) with others techniques.
The APA function returns the current value of the specified cell considering a factor (0.025 in this case) used with the Hodgkin-Huxley differential equations. This factor is needed to indicate the relation with the delay time of the CD++ rule (because the minimum time that we can indicate on a CD++ rule is 1 ms and this function need less than 1 ms. This is explained two paragraphs above).
To identify the tissue and use the internal values of the function, the x-y position of the cell (which represents the tissue of the heart in the DEVS model) is indicated to the function as arguments.
These five arguments are packed as follows:
Argument 1 = coor(x) * 10000000 + coor(y) * 100000 + NeighborsActiveIndicator * 10000 + DiscretizationFactor.

The discretization factor (0.025 on this model) is used to indicate function APA the milliseconds used to discretize the values with the differential equations of Hodgkin-Huxley formula. The minimum unit time we can use in CD++ is 1 millisecond and the Hodgkin-Huxley implementation changes the tissue values every 0.005 milliseconds. This is the reason a scale was needed for the simulation. The scale in use is 1 CD++ ms = 0.005 Heart ms. All in all, to discretize every 0.025ms, CD++ uses a rule delay of 5ms (0.005 * 5 = 0.025ms).
The second arguments it's only the current cell value (voltage) of the tissue (**Argument 2)**
In the APA function (implemented in CD++) these arguments are unpacked.

An example of the cells voltages values generated by a simulation without quantum is showed above.

```
Line : 83 - Time: 00:00:00:000
               0          1          2          3          4
    +-------------------------------------------------------------+
   0|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   1|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   2|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   3|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   4|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
    +-------------------------------------------------------------+

Line : 212 - Time: 00:00:00:005
    +-------------------------------------------------------------+
   0|   -81.59893   -81.59893   -81.59893   -81.59893   -81.59893|
   1|   -81.59893   -81.59893   -81.59893   -81.59893   -81.59893|
   2|   -81.59893   -81.59893   -81.59893   -81.59893   -81.59893|
   3|   -81.59893   -81.59893   -81.59893   -81.59893   -81.59893|
   4|   -81.59893   -81.59893   -81.59893   -81.59893   -81.59893|
    +-------------------------------------------------------------+
                      ...
Line : 8984 - Time: 00:00:00:345
    +-------------------------------------------------------------+
```

```
   0|    -1.00816    -1.00816    -1.00816    -1.00816    -1.00816|
   1|    -1.00816    -1.00816    -1.00816    -1.00816    -1.00816|
   2|    -1.00816    -1.00816    -1.00816    -1.00816    -1.00816|
   3|    -1.00816    -1.00816    -1.00816    -1.00816    -1.00816|
   4|    -1.00816    -1.00816    -1.00816    -1.00816    -1.00816|
    +------------------------------------------------------------+
                         ...
Line : 12467 - Time: 00:00:00:480
    +------------------------------------------------------------+
   0|    23.58846    23.58846    23.58846    23.58846    23.58846|
   1|    23.58846    23.58846    23.58846    23.58846    23.58846|
   2|    23.58846    23.58846    23.58846    23.58846    23.58846|
   3|    23.58846    23.58846    23.58846    23.58846    23.58846|
   4|    23.58846    23.58846    23.58846    23.58846    23.58846|
    +------------------------------------------------------------+
                         ...
Line : 31043 - Time: 00:00:01:200
    +------------------------------------------------------------+
   0|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   1|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   2|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   3|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   4|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
    +------------------------------------------------------------+
…
Line : 169460 - Time: 00:00:06:565
    +------------------------------------------------------------+
   0|   -14.50695   -14.50695   -14.50695   -14.50695   -14.50695|
   1|   -14.50695   -14.50695   -14.50695   -14.50695   -14.50695|
   2|   -14.50695   -14.50695   -14.50695   -14.50695   -14.50695|
   3|   -14.50695   -14.50695   -14.50695   -14.50695   -14.50695|
   4|   -14.50695   -14.50695   -14.50695   -14.50695   -14.50695|
    +------------------------------------------------------------+
                         ...
Line : 1988876 - Time: 00:01:17:085
    +------------------------------------------------------------+
   0|   -78.49942   -78.49942   -78.49942   -78.49942   -78.49942|
   1|   -78.49942   -78.49942   -78.49942   -78.49942   -78.49942|
   2|   -78.49942   -78.49942   -78.49942   -78.49942   -78.49942|
   3|   -78.49942   -78.49942   -78.49942   -78.49942   -78.49942|
   4|   -78.49942   -78.49942   -78.49942   -78.49942   -78.49942|
    +------------------------------------------------------------+

Line : 1989005 - Time: 00:01:17:090
    +------------------------------------------------------------+
   0|   -78.49984   -78.49984   -78.49984   -78.49984   -78.49984|
   1|   -78.49984   -78.49984   -78.49984   -78.49984   -78.49984|
   2|   -78.49984   -78.49984   -78.49984   -78.49984   -78.49984|
   3|   -78.49984   -78.49984   -78.49984   -78.49984   -78.49984|
   4|   -78.49984   -78.49984   -78.49984   -78.49984   -78.49984|
    +------------------------------------------------------------+

Line : 1989134 - Time: 00:01:17:095
    +------------------------------------------------------------+
   0|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   1|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   2|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   3|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   4|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
    +------------------------------------------------------------+
```

Implementation for Q-DEVS

This implementation is the Adaptive Quantization [13] implementation for the Heart model.
The function used for local updating is the inverse function of the original (APA), which receives an additional argument indicating the quantum value for this technique.
This function returns two values, which are:

1) the next value will overcome the LastThresholdValue (the value that changes the region) for the analyzed cell according to the specified quantum.
2) the time when this will occur.

With this function, we can know when a cell will pass the threshold value with the indicated quantum and which is that value.
However, the function cannot return two values, so a new implementation mechanism was used in order to analyze the behavior of this quantum type and maintain low the complexity when designing models. The implemented mechanism used here for the arguments is like a BEAN technique, where all the necessary input values are set on the object to use before of calling the function and the resultant values are got after calling the function from the used object.
This was made using a setArg and getArg new functions of CD++.

```
12  initialvalue : -85.0
13  initialCellsValue : QCorazon.val
14  localtransition : corazon-rule

15  [corazon-rule]
%La funcion setArg es para setear parametros en una celda.
%La funcion getArg es para obtener parametros de una celda.
%Ambas reciben las coordenadas de la celda y el nro de parametro en cuetion.

%Parámetros:
%0)Voltaje actual de la celda
%1)Estado de la celda (reposo o activa)
%2)Factor de discretizacion de la funcion original APA
%3)Quantum utilizado (10 en este ejemplo)
%Luego se setea otro parametro que es el delay para
%llegar al proximo q que es lo que devuelve APAINV
%y la funcion APAINV setea otro parametro mas que es
%el valor que toma la funcion APA en el tiempo calculado (luego del delay)

%La siguiente regla, que ejecuta 1ro la condicion (true en este caso),
%Luego el delay (que es cuando calculo el delay hasta el proximo q y
%la funcion internamente setea el voltaje con el que llego al proximno q
%y luego actualiza el valor de la celda con el parametro 5
%seteado por la funcion APAINV


16  rule : { getArg(cellpos(0)*100000+cellpos(1)*100+5) }
    {
        setArg(cellpos(0)*100000+cellpos(1)*100+0,(0,0))+
        setArg(cellpos(0)*100000+cellpos(1)*100+1, if((0,0)=-83.0,1.0,0.0) )+
        setArg(cellpos(0)*100000+cellpos(1)*100+2,0.025)+
        setArg(cellpos(0)*100000+cellpos(1)*100+3,10)+
        setArg(cellpos(0)*100000+cellpos(1)*100+4, APAINV(cellpos(0),cellpos(1)))+
        getArg(cellpos(0)*100000+cellpos(1)*100+4) / 0.025 * 5
    } { t }
```

The model description is the same as in the original heart model except for the rule, which uses the inverse function of the APA function (called APAINV).
The delay is obtained with the use of getArg(4) function and the new cell voltage is obtained with getArg(5) function (The getArg function gets internal values calculated with the local function).
When this rule is evaluated, it sets the delay for the analyzed cell with the future time when next region or threshold will be achieved (when the new value will be transmitted through the CD++ tool).
The arguments used on the APAINV function are the same of the APA function plus a new argument, the quantum, because with this mechanism the quantum value is internal (it is for the APAINV function, not for CD++). Thus this quantum value is not used on the CD++ tool, it is used on the inverse function APAINV, as an internal argument. This is an internal quantifier and CD++ makes no operations with this specified quantum. Quantification is given by the local function not by the tool. Of course, if we try to quantify this function also with the tool, no advantages will be obtained because of the values used would be already quantified internally with the function, which is returning the future values setting the corresponding future time for each value.
As we can see, the discretization factor used in APAINV is the same as in the APA function, but the delay time is not (it depends on the result of the APAINV evaluation). The discretization factor is used always on the original function (for the discretization of the Hodgkin and Huxley differential equations).
The APAINV function uses APA internal calculations to calculate each value. In a way is searching the next value that will change its region.
An example of the cells voltages values generated by a simulation with quantum 20 is showed above.

```
Line : 83 - Time: 00:00:00:000
               0          1          2          3          4
    +-------------------------------------------------------------+
   0|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   1|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
```

```
   2|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   3|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   4|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
    +-------------------------------------------------------------+

Line : 212 - Time: 00:00:00:020
    +-------------------------------------------------------------+
   0|   -79.80072   -79.80072   -79.80072   -79.80072   -79.80072|
   1|   -79.80072   -79.80072   -79.80072   -79.80072   -79.80072|
   2|   -79.80072   -79.80072   -79.80072   -79.80072   -79.80072|
   3|   -79.80072   -79.80072   -79.80072   -79.80072   -79.80072|
   4|   -79.80072   -79.80072   -79.80072   -79.80072   -79.80072|
    +-------------------------------------------------------------+

                              ...
Line : 728 - Time: 00:00:00:350
    +-------------------------------------------------------------+
   0|     4.28287     4.28287     4.28287     4.28287     4.28287|
   1|     4.28287     4.28287     4.28287     4.28287     4.28287|
   2|     4.28287     4.28287     4.28287     4.28287     4.28287|
   3|     4.28287     4.28287     4.28287     4.28287     4.28287|
   4|     4.28287     4.28287     4.28287     4.28287     4.28287|
    +-------------------------------------------------------------+

Line : 857 - Time: 00:00:00:380
    +-------------------------------------------------------------+
   0|    20.51040    20.51040    20.51040    20.51040    20.51040|
   1|    20.51040    20.51040    20.51040    20.51040    20.51040|
   2|    20.51040    20.51040    20.51040    20.51040    20.51040|
   3|    20.51040    20.51040    20.51040    20.51040    20.51040|
   4|    20.51040    20.51040    20.51040    20.51040    20.51040|
    +-------------------------------------------------------------+

                             ...
Line : 1115 - Time: 00:00:01:200
    +-------------------------------------------------------------+
   0|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   1|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   2|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   3|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
   4|    -0.04211    -0.04211    -0.04211    -0.04211    -0.04211|
    +-------------------------------------------------------------+

Line : 1244 - Time: 00:00:18:430
    +-------------------------------------------------------------+
   0|   -20.00027   -20.00027   -20.00027   -20.00027   -20.00027|
   1|   -20.00027   -20.00027   -20.00027   -20.00027   -20.00027|
   2|   -20.00027   -20.00027   -20.00027   -20.00027   -20.00027|
   3|   -20.00027   -20.00027   -20.00027   -20.00027   -20.00027|
   4|   -20.00027   -20.00027   -20.00027   -20.00027   -20.00027|
    +-------------------------------------------------------------+

                              ...
Line : 1631 - Time: 00:01:17:095
    +-------------------------------------------------------------+
   0|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   1|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   2|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   3|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   4|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
    +-------------------------------------------------------------+
```

As we can see on the above lines, at Time: 00:00:01:200 we have the value “-0.04211” and the next output generated by the tool is at Time: 00:00:18:430 where the value is “-20.00027” because of the quantum value “20” in use. This is a very high time because we have saved 17 seconds with a default delays of 5 ms, which means 16995ms / 5ms = 3399 outputs avoided.

Implementation for GDEVS

[7] This implementation is based on polynomial approximations of the original function.
The first step in this study was to find a polynomial approximation to the original PDE (Partial Differential Equations) defining the cell's behavior. The following Figure shows the result of this approximation function:

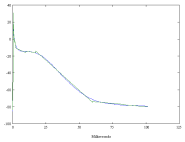


Figure 11. Linear approximation of the Action Potential function.

We approximated the initial equation experimental data using twelve polynomials of degree one, in order to built a G-DEVS model of order 1. A higher level of accuracy can be obtained using G-DEVS of higher level with the same number of states and events to treat. In the present case, the identification of the parameters in each of the polynomials was obtained minimizing a quadratic criterion using minimum squares. The polynomials we used in Figure 11 are defined by:

$P_i(t) = a_i t + b_i \qquad \forall i \in [1, 12]$

using the following coefficients:

i	ai	bi	Time (ms)
1	0.12686423	-83.0275035	[0, 285)
2	0.36919131	-151.270449	[285, 315)
3	1.09229907	-378.594614	[315, 360)
4	0.15268376	-38.3409317	[360, 415)
5	-0.00494316	26.3	[415, 470)
6	-0.03836983	40.7043237	[470, 980)
7	-0.01071088	13.0643535	[980, 1780)
8	-0.001545	-3.18042127	[1780, 5030)
9	-0.00036934	-11.4659057	[5030, 18430)
10	-0.0019262	16.6510351	[18430, 39940)
11	-0.00045075	-47.0156891	[39940, 77095)
12	0	-83	[77095, inf)

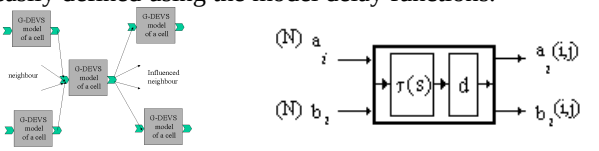
Table 1. Polynomial coefficients for the action potential model

Even the original function has an appearance to be simple, we needed to use twelve polynomials. This was due to the fact that, when the cell is triggered the signal generated by the Hodgkin-Huxley model is highly non-linear. Thus, we needed to approximate the action potential using different polynomials (as shown in Table 1).

Figure 12. Approximation of the Action Potential function: action triggering.

When using GDEVS for this model, we need to transform the coefficients in the polynomials into discrete event signals. Each cell uses polynomial coefficients to compute the current state and to inform the cell's state to the neighbors as showed in Figure 13. The specification of the local computing function included in each of the cells will now receive the current coefficient from the neighboring cells, as shown in figure 13.a). Each cell is now defined as shown in figure 13.b), and it will receive the coefficients of the neighbors by the way of an event of order 1, which will be used to compute the state of the cell. The cell's

outputs will now be the current cell states specified as polynomial coefficients. Timing of activation for each polynomial can be easily defined using the model delay functions.



(a) (b)

Figure 13. GDEVS cell specification (a) model interconnection (b) cell input data.

Using these ideas and the polynomial definitions in Table 1, we can now define the actions of each of the cell's local computing functions, which are described by the state graph in the following Figure 14. The figure defines a G-DEVS model with the classical state transition functions using events of order 1. The figure represents internal transitions in dotted lines, and external transitions in full lines.

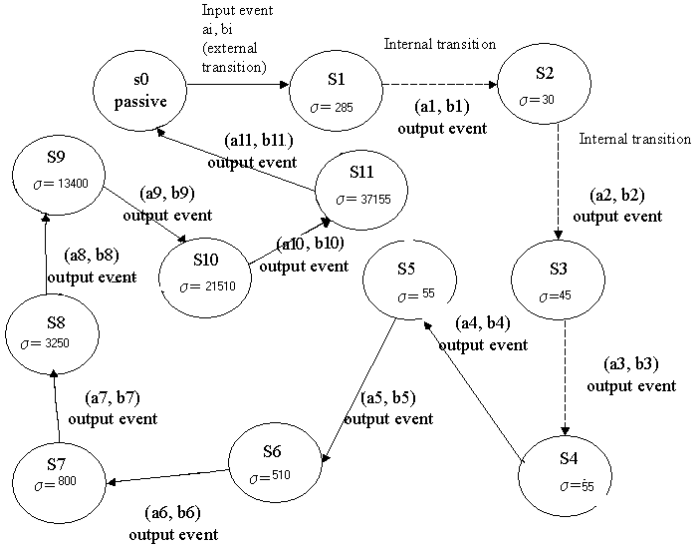


Figure 14. GDEVS specification of a cell.

As we see, the cell is inactive until it receives an external stimulus from a neighboring cell. In that case, the cell is activated, and it produces internal state changes, (represented by the coefficient in the polynomials, which are transmitted to the neighboring cells after the delay). The model flows through eight different states represented by each of the polynomials, plus an extra state to put the model into a resting state.

This specification will generate an output trajectory as the one described by the linear approximation in Figure 12. As we can see, this highly improves model precision at a low cost, in terms of both execution time and ease of modeling. The following Figure 15 shows the model implementation for a cell space in which each cell is created using the state machine in Figure 14.

```
%modelo del Corazon implementado con GDEVS en 2d con multiples reglas
%para los coeficientes
%-----------------------------------------------------------------
%Ejecutar con parametros -c00:00:00:005 (discrete time tik 5ms)
%-----------------------------------------------------------------

1   [top]
2   components : heart-GDEVS

3   [heart-GDEVS]
4   type : cell
5   dim : (5,5)
6   delay : transport
7   defaultDelayTime  : 5
8   border : nowrapped
9   neighbors : (0,-1) (0,0) (-1,0) (-1,-1)
```

```
10  neighbors : (0,1)  (1,0) (-1,1) (1,1) (1,-1)
11  initialvalue : -83.0
12  initialCellsValue : corazon2dnr.val
13  localtransition : heart-rule-GDEVS

14  [heart-rule-GDEVS]

15 rule : { -85.0 } { 0 + setArg(cellpos(0)*100000+cellpos(1)*100 + 0, time) } { (0,0) =
    -83.0 }
16 rule : { 0.126864227498678 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100 + 0) )
    - 83.027503469064 } 5 { (0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 285 }
17 rule : { 0.369191314285707 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100 + 0) )
    - 151.270449333331 } 285 { (0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 285 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 315 }
18 rule :      { 1.09229907142865  * ( time - getArg(cellpos(0)*100000+cellpos(1)*100 +
    0) ) - 378.594613571453 } 30 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 315 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 360 }
19 rule :      { 0.152683762237757 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100 +
    0) ) - 38.3409316783197 } 45 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 360 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 415 }
20 rule :      { -0.00494316363636244 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100
    + 0) ) + 26.3 } 55 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 415 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 470 }
21 rule :      { -0.0383698302729355 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100
    + 0) ) + 40.7043236511447 } 45 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 470 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 980 }
22 rule :      { -0.010710876279456 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100 +
    0) ) + 13.0643535113489 } 510 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 980 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 1780 }

23 rule :      { -0.001545 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100 + 0) ) -
    3.18042127392896 } 800 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 1780 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 5030 }
24 rule :      { -0.000369342915756289 * ( time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) ) - 11.4659056569606 } 3250 {(0,0) != -
    83.0 and time - getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 5030 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 18430 }
25 rule :      { -0.00192620311952202 * ( time - getArg(cellpos(0)*100000+cellpos(1)*100
    + 0) ) + 16.6510350787401 } 13400 {(0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 18430 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 39940 }
26 rule :      { -0.000450748899007899 * ( time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) ) - 47.0156891246265 } 21505 {(0,0) !=
    -83.0 and time - getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 39940 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) < 77095 }
27 rule :      { -83.0 } 37155 { (0,0) != -83.0 and time -
    getArg(cellpos(0)*100000+cellpos(1)*100 + 0) >= 77095 }
28 rule :      { (0,0) } 5 { t }
```

Figure 15. Cell-DEVS/GDEVS implementation of the heart tissue model.

This specification starts by defining the size of the cell space (5 x 5), and the remaining parameters needed by a Cell-DEVS specification. In this case, transport delays, a non-wrapped model, and the neighborhood shape, which includes all the adjacent cells. Then, we define the local computing function, heart-rule-GDEVS. This local computing function follows the specification in Figure 14. If a stimulus is received when the cell is inactive ((0,0)=-83), it will change to the corresponding state (Si, to the left of the specification). Each of the rules represents a cell's state change, and the spread of the coefficients to the neighbors. Each of the cells will repeat the behavior here defined, while storing the voltage value for display, which is showed in the following figure.

```
Line : 212 - Time: 00:00:00:000
                0           1           2           3           4
    +-------------------------------------------------------------+
```

```
   0|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   1|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   2|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   3|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
   4|   -85.00000   -85.00000   -85.00000   -85.00000   -85.00000|
    +------------------------------------------------------------+
Line : 341 - Time: 00:00:00:005
    +------------------------------------------------------------+
   0|   -83.02750   -83.02750   -83.02750   -83.02750   -83.02750|
   1|   -83.02750   -83.02750   -83.02750   -83.02750   -83.02750|
   2|   -83.02750   -83.02750   -83.02750   -83.02750   -83.02750|
   3|   -83.02750   -83.02750   -83.02750   -83.02750   -83.02750|
   4|   -83.02750   -83.02750   -83.02750   -83.02750   -83.02750|
    +------------------------------------------------------------+
Line : 7565 - Time: 00:00:00:285
    +------------------------------------------------------------+
   0|   -47.50552   -47.50552   -47.50552   -47.50552   -47.50552|
   1|   -47.50552   -47.50552   -47.50552   -47.50552   -47.50552|
   2|   -47.50552   -47.50552   -47.50552   -47.50552   -47.50552|
   3|   -47.50552   -47.50552   -47.50552   -47.50552   -47.50552|
   4|   -47.50552   -47.50552   -47.50552   -47.50552   -47.50552|
    +------------------------------------------------------------+
Line : 7823 - Time: 00:00:00:615
    +------------------------------------------------------------+
   0|    18.83352    18.83352    18.83352    18.83352    18.83352|
   1|    18.83352    18.83352    18.83352    18.83352    18.83352|
   2|    18.83352    18.83352    18.83352    18.83352    18.83352|
   3|    18.83352    18.83352    18.83352    18.83352    18.83352|
   4|    18.83352    18.83352    18.83352    18.83352    18.83352|
    +------------------------------------------------------------+
Line : 8855 - Time: 00:00:00:975
    +------------------------------------------------------------+
   0|     5.02038     5.02038     5.02038     5.02038     5.02038|
   1|     5.02038     5.02038     5.02038     5.02038     5.02038|
   2|     5.02038     5.02038     5.02038     5.02038     5.02038|
   3|     5.02038     5.02038     5.02038     5.02038     5.02038|
   4|     5.02038     5.02038     5.02038     5.02038     5.02038|
    +------------------------------------------------------------+
                                    …
Line : 10763 - Time: 00:01:09:795
    +------------------------------------------------------------+
   0|   -68.78235   -68.78235   -68.78235   -68.78235   -68.78235|
   1|   -68.78235   -68.78235   -68.78235   -68.78235   -68.78235|
   2|   -68.78235   -68.78235   -68.78235   -68.78235   -68.78235|
   3|   -68.78235   -68.78235   -68.78235   -68.78235   -68.78235|
   4|   -68.78235   -68.78235   -68.78235   -68.78235   -68.78235|
    +------------------------------------------------------------+
                                    …
```

As we can see, outputs are generated at the specified times for each polynomial representation showed on figure 14. For instance, after Time: 00:00:00:005 with value "-83.02750" next activation occurs at Time: 00:00:00:285 with value "-47.50552". Time = 5ms correspond to state S0 and Time = 285ms correspond to state S1 of Figure 14.

5 Watershed Model Description and Implementations

[16] In this section we considered a case of study of a natural phenomena's. Watershed are natural regions defined by the shape of the land surface, which store up water because of rain, mountains ice melting and rivers (as is shown in Figure 16).
With the use of computerized Geographic Information Systems (GIS) the watershed limits can be defined, generating topology maps of the zone with high precision and resolution

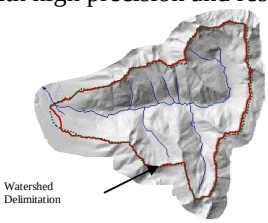


Figure 16 – Watershed GIS System View

Several verticals layers compose a watershed : air, vegetation, water surface, land surface and stones. The model can be divided in equal portions of land (cells) and a hydrology theory model can be applied to each portion [15] **.** In each cell the behavior of the water flow (how the water is distributed through the land and how the topology of the land makes it concentrate or accumulate in determined places) can be analyzed.

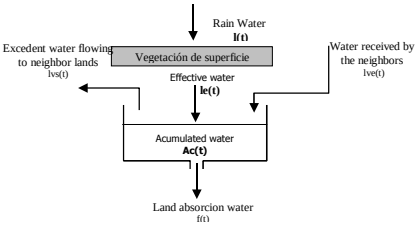


Figure 17 – Hydrology theory model

Figure 17 shows the theory hydrologic model used for the simulation. When the waterfall part of it is absorbed by the vegetation existing in the surface, the rest, the effective quantity of water, is received by the surface. Depending on the topology of the land, the cells can also receive/send, water from/to the neighbor cells. In this case, the neighbor cells are the 4 ones located in the North, South, East and West. At the end , part of the water received by the surface is lost due to the filtration over the land and stones.
Therefore, it can be defined over this model the depth of the water during a period of time in one cell through:

$$Ac(t) = \int_0^t (le(t) + \sum_{i=1}^{4} lve_i(t) - \sum_{i=1}^{4} lvs_i(t) - f(t))dt \qquad (1)$$

$$lvs_i(t) = \frac{R_s C_i(t) Ac(t)}{D_i} \qquad (2)$$

Where:
Ac = Accumulated water in one cell in millimeters
le = Quantity of effective water
lve$_i$ = Quantity of water dumped from the neighbor “i” to the cell
lvs$_i$ = Quantity of water dumped to the neighbor “i” when the limit of the cell is exceed.
f = Quantity of filtrated water.
R$_s$ = Surface characteristic (rugosity)
D$_i$ = distance between the cell and the ith neighbor
C$_i$ = land inclination with respect to the ith neighbor, calculated as follows:

$$C(t) = \frac{Ac(t) + h - Ac_i(t) - h_i}{D_i} \qquad (3)$$

Where:
Ac = accumulated water on the cell
h = cell high
Ac$_i$ = accumulated water on the cell i
h$_i$ = high of the cell i

As we can see in the equation (1), the accumulated quantity of water during a period of time depends on:

- The quantity of effective water (rain).
- The quantity of water dumped from the neighbor cells (depending on the quantity of effective rain water plus the water received --depending on the land inclination-- by the neighbor cells minus the water sent to his neighbors when the capability of the cell is achieved, and minus the filtered water on the stones and the land).

The equation (2) defines the quantity of water sent to the neighbors, the argument **C$_i$** (equation (3)) determines the quantity depending on the land inclination, which is the height difference between the cell and its neighbors.

The simulation is the height of water on each cell through a period of time with an intensity of 7,92 mm per hour, rain defined as very intense, constant through that period of time and in all the land surface.

Original CD++ Implementation

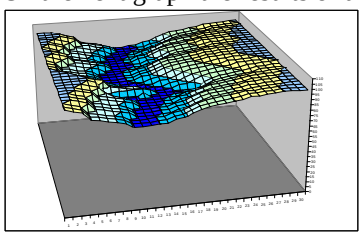
```
01      [top]
02      components : Watershed
03
04      [Watershed]
05      type : cell
06      dim : (30,30,2)
07      delay : transport
08      defaultDelayTime : 1000
09      border : nowrapped
10      neighbors :                   Watershed(-1,0,0)
11      neighbors : Watershed(0,-1,0) Watershed(0,0,0) Watershed(0,1,0)
12      neighbors :                   Watershed(1,0,0)
13
14      neighbors :                   Watershed(-1,0,1)
15      neighbors : Watershed(0,-1,1) Watershed(0,0,1) Watershed(0,1,1)
16      neighbors :                   Watershed(1,0,1)
17
18      initialValue : 0
19      initialCellsValue : topo_I.val
20      localtransition : Hydrology
21
22      [Hydrology]
23
24      rule : {0.0022 + (0,0,0) - if((((-1,0,0) != ?) and (((0,0,1) +
        (0,0,0))>((-1,0,1) + (-1,0,0)))),(((((0,0,0) + (0,0,1) - (-1,0,0) - (-
        1,0,1))/1000) * (0,0,0))/1000),0) - if((((1,0,0) != ?) and (((0,0,1) +
        (0,0,0))>((1,0,1)  +  (1,0,0)))),(((((0,0,0) + (0,0,1) - (1,0,0) -
        (1,0,1))/1000) * (0,0,0))/1000),0) - if((((0,-1,0) != ?) and (((0,0,1) +
        (0,0,0))>((0,-1,1)+(0,-1,0)))),(((((0,0,0) + (0,0,1) - (0,-1,0) - (0,-
        1,1))/1000) * (0,0,0))/1000),0) - if((((0,1,0) != ?) and (((0,0,1) +
        (0,0,0))>((0,1,1) + (0,1,0)))),(((((0,0,0) + (0,0,1) - (0,1,0) -
        (0,1,1))/1000) * (0,0,0))/1000),0) + if((((-1,0,0) != ?) and (((-1,0,1)
        + (-1,0,0))>((0,0,1) + (0,0,0)))),((((-1,0,0) + (-1,0,1) - (0,0,0) -
        (0,0,1)) * (-1,0,0))/1000),0) + if((((1,0,0) != ?) and (((1,0,1) +
        (1,0,0))>((0,0,1) + (0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) - (0,0,1))
        * (1,0,0))/1000),0) + if((((0,-1,0) != ?) and (((0,-1,1) + (0,-
        1,0))>((0,0,1) + (0,0,0)))),((((0,-1,0) + (0,-1,1) - (0,0,0) - (0,0,1))
        * (0,-1,0))/1000),0) + if((((0,1,0) != ?) and (((0,1,1) +
        (0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) + (0,1,1) - (0,0,0) - (0,0,1))
        * (0,1,0))/1000),0) } 1000 { cellpos(2)=0 }

25      rule : { (0,0,0) } 1000 { t }
```

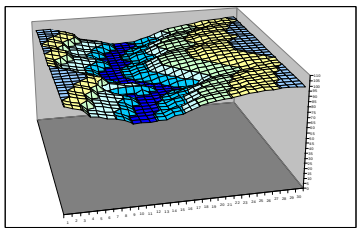
Figure 18 – Watershed model CD++ implementation

Figure 18 shows the model specification of the original model.
Line 2 defines the cell-DEVS component denominated Watershed.
Line 4 defines the component. **Line 5** indicates CD++ that is a cellular model. The model has two slides, slide 0 represents the accumulated water and slide 1 the height of each cell with respect to the sea level.
Line 6 defines de dimension. Each slide has 30x30 cells and each one of them represents 1mx1m.
Line 7 defines the delay type as transport. **Line 8** defines de default delay as 1 second.
Line 9 defines the border type for the model, which is no wrapped, which means that the border cells are not connected with the opposite border. **Lines 10 to 16** defines the relative positions of the neighbors which are all the adjacent top, bottom, left and right cells and the analyzed cell (this means North, South, East and West because this model considers water interchange between these cells).
Line 18 defines the default value for every cell, which is 0.0. The real initial value of each cell will be defined on next line. **Line 19** indicates the file with the initial values for each cell, which in this case is the initial capability of each cell. **Line 20** is the name of the section with the behavior rules for the cellular model. **Line 22** is beginning of the rules description section. It exists two ruled, because the behavior is the same for all cells on slide 0 and 1. **Line 24** defines the quantity of water will have a cell after 1 second, considering the water it had before, the water increment because of rain on the period of time, the water added by the neighbors who achieved its capabilities and the water sent to the neighbors if this cell achieved its maximum capability. The sent and received water depend of the maximum

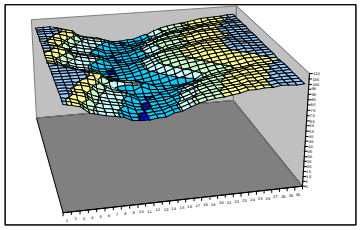
capability of each cell and the land inclination. Equation (3) calculates individually with each neighbor cell the land inclination. If a neighbor has more water than the height of the analyzed cell, the cell will receive water from it once the neighbor level is achieved. The same calculation is made if the height of the analyzed cell is bigger than the neighbor height.
The rule of line 24 starts with the constant value, which represents the quantity of effective water of an intense rain (0.0022 mm/sec = 7.92 mm/hour). This value is added to the water the cell already had a second before, plus the received water from its neighbors, minus the water sent to its neighbors.
Line 25 does not modify the state of the cells.
On the next graph the results of a simulation can be observed considering an intense rain.



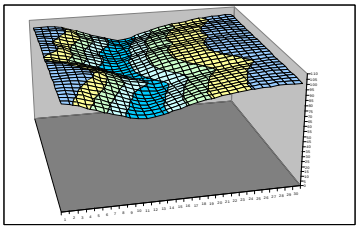
(a) Land topology.



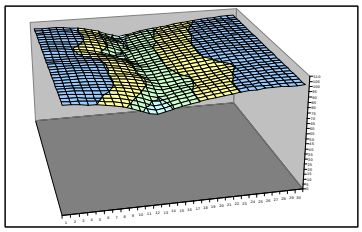
(b) After 5 minutes of rain.



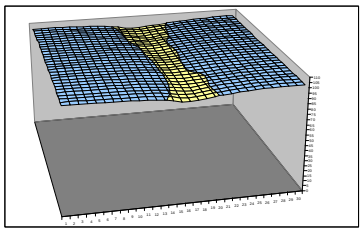
(c) After 10 minutes.



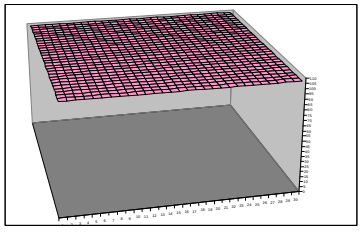
(d) After 15 minutes.



(e) After 20 minutes.



(f) After 30 minutes.



(g) After 45 minutes of rain the land is fully filled with the water.

New CD++ Implementation

This is the Watershed model simplified with the function WSHED incorporated into CD++ (same mechanism as mentioned in the Heart model section, with the APA function).
As a difference of the original model, this model implementation uses only one plane (2 dimensions) because cells capabilities are stored automatically on the internal WSHED function at time 0. In the original model, one plane was used to store the capability of each cell and on the other the accumulation of water. With this implementation, the capability and the variation are the same as a result of the sum of both (capability and variation). The second plane used before to keep alive the model is no more needed because of the new CD++ "-c" argument, which keeps the model alive generating external messages for those cells without activity.

```
1.  [top]
```

```
2.  components : Watershed

3.  [Watershed]
4.  type : cell
5.  dim : (16,28)
6.  delay : transport
7.  defaultDelayTime : 1000
8.  border : nowrapped
9.  neighbors :                     Watershed(-1,0)
10. neighbors :   Watershed(0,-1) Watershed(0,0) Watershed(0,1)
11. neighbors :                     Watershed(1,0)
12. initialvalue : 0.0
13. initialCellsValue : topo1528.val
14. localtransition : Hydrology

15. [Hydrology]
% La funcion WSHED recibe como parametros la celda, su propio valor actual, la celda
vecina y el valor de ésta.
%Esto se repite para cada una de las vecinas (el desplazamiento para representar las
coordenadas de las vecinas, se indica con los sig nros: %0=>1, 1=>2, -1=>3).

16. rule: { (0,0) + 0.0022 +
    if( ((-1,0) != ?), WSHED( cellpos(0)*1000+cellpos(1)*1000000+(0,0),
    3*1000+1*1000000+(-1,0) ), 0.0) +

    if( ((1,0) != ?), WSHED( cellpos(0)*1000+cellpos(1)*1000000+(0,0),
    2*1000+1*1000000+(1,0) ), 0.0) +

    if( ((0,1) != ?), WSHED( cellpos(0)*1000+cellpos(1)*1000000+(0,0),
    1*1000+2*1000000+(0,1) ), 0.0) +

    if( ((0,-1) != ?), WSHED( cellpos(0)*1000+cellpos(1)*1000000+(0,0),
    1*1000+3*1000000+(0,-1) ), 0.0)
    } 1000 { t }
```

The function WSHED is called three times in the same rule in order to calculate the new value considering the four neighbors. The function receives 6 arguments, which are:

- The x and y position of the analyzed cell (2 arguments)
- The current value (accumulated water) of the cell
- The x and y offset position of its neighbor to indicate the function which neighbor is sending water to the cell (because the function has to calculate the amount of water to receive from each neighbor) (2 arguments).
- The current value (accumulated water) of the analyzed neighbor.

On **Lines 1 and 2** the unique Cell-DEVS model Watershed is defined. On **line 3** the model description begins. **Line 4** indicates CD++ that is a cellular model. **Line 5** is the defined dimension of the cellular space. In this case, we have a space of 16x28 cells. L**ine 6** defines the delay type of the model for each cell. In this case, the delay type used is transport. **Line 7** defines default delay, 1000 milliseconds in this model and it's the same delay for every cell (this is not necessary true on the real model, but is enough to make the simulation analysis). **Line 8** defines the border type for the model, which is no wrapped, which means that the border cells are not connected with the opposite border. **Lines 9 to 11** defines the relative positions of the neighbors which are all the adjacent top, bottom, left and right cells and the analyzed cell (this means North, South, East and West because this model considers water interchange between these cells). **Line 12** defines the default value for every cell, which is 0.0. The specific initial value of each cell will be defined on next line. **Line 13** indicates the file with the initial values for each cell, which in this case is the initial capability of each cell. **Line 14** is the name of the section with the behavior rules for the cellular model. **Line 15** is the beginning of the rules description section. **Line 16** is the first rule (and the unique in this model). This rule defines the local updating function of each cell, which represents a portion of the land.

An example of the cells values (capability and variation) generated by a simulation without quantum is showed above.

```
Line : 1352 - Time: 00:00:00:000
              0           1           2           3
    +-----------------------------------------------
   0|   100.00000    98.00000    96.00000    89.00000
   1|   100.00000    98.00000    96.00000    89.00000
   2|   100.00000    98.00000    96.00000    89.00000
   3|   100.00000   100.00000    98.00000    96.00000
```

```
   4|   100.00000   100.00000    98.00000    96.00000
   5|   100.00000   100.00000    98.00000    96.00000
   6|   100.00000   100.00000   100.00000    98.00000
   7|   100.00000   100.00000   100.00000    98.00000
   8|   100.00000   100.00000   100.00000   100.00000
   9|   100.00000   100.00000   100.00000   100.00000
  10|   100.00000   100.00000   100.00000   100.00000
  11|   100.00000   100.00000   100.00000   100.00000
  12|   100.00000   100.00000   100.00000    98.00000
  13|   100.00000   100.00000    98.00000    96.00000
  14|   100.00000    98.00000    96.00000    89.00000
  15|    98.00000    96.00000    89.00000    88.00000
    +-------------------------------------------------

             …
Line : 270632 - Time: 00:02:00:000
    +-------------------------------------------------
   0|   100.26397    98.29514    96.29774    89.38235
   1|   100.26397    98.29542    96.29808    89.38402
   2|   100.26397    98.32582    96.33368    89.50832
   3|   100.26400   100.26394    98.29510    96.29762
   4|   100.26400   100.26397    98.29542    96.29808
   5|   100.26400   100.26397    98.32582    96.33369
   6|   100.26400   100.26400   100.26394    98.29538
   7|   100.26400   100.26400   100.26397    98.32582
   8|   100.26400   100.26400   100.26400   100.26394
   9|   100.26400   100.26400   100.26400   100.26400
  10|   100.26400   100.26400   100.26400   100.26400
  11|   100.26400   100.26400   100.26400   100.26394
  12|   100.26400   100.26400   100.26394    98.32578
  13|   100.26400   100.26394    98.32578    96.33621
  14|   100.26394    98.32578    96.33621    89.51895
  15|    98.29513    96.33368    89.51847    88.32010
    +-------------------------------------------------

             …
Line : 674552 - Time: 00:05:00:000
    +-------------------------------------------------
   0|   100.65981    98.84815    96.89319    90.47827
   1|   100.65981    98.85942    96.91145    90.58905
   2|   100.65983    99.01795    97.14164    91.34087
   3|   100.66000   100.65966    98.84791    96.89247
   4|   100.66000   100.65982    98.85942    96.91153
   5|   100.66000   100.65984    99.01796    97.14321
   6|   100.66000   100.66000   100.65966    98.85919
   7|   100.66000   100.66000   100.65984    99.01796
   8|   100.66000   100.66000   100.66000   100.65968
   9|   100.66000   100.66000   100.66000   100.66000
  10|   100.66000   100.66000   100.66000   100.66000
  11|   100.66000   100.66000   100.66000   100.65968
  12|   100.66000   100.66000   100.65968    99.01770
  13|   100.66000   100.65968    99.01770    97.18155
  14|   100.65965    99.01769    97.18155    91.52604
  15|    98.84766    97.14160    91.50746    89.54809
    +-------------------------------------------------

             …
Line : 1347302 - Time: 00:10:00:000
    +-------------------------------------------------
   0|   101.31935    99.99552    98.37088    94.30018
   1|   101.31939   100.10094    98.60284    95.22174
   2|   101.31956   100.41216    99.12812    96.26514
   3|   101.32000   101.31911    99.99493    98.37058
   4|   101.32000   101.31950   100.10097    98.61098
   5|   101.32000   101.31964   100.41219    99.15937
   6|   101.32000   101.32000   101.31915   100.10040
   7|   101.32000   101.32000   101.31964   100.41228
   8|   101.32000   101.32000   101.32000   101.31928
   9|   101.32000   101.32000   101.32000   101.32000
  10|   101.32000   101.32000   101.32000   101.32000
  11|   101.32000   101.32000   101.32000   101.31928
  12|   101.32000   101.32000   101.31928   100.41155
  13|   101.32000   101.31928   100.41155    99.31909
  14|   101.31903   100.41143    99.31902    97.05556
  15|    99.97222    99.12374    96.93607    95.74600
    +-------------------------------------------------
```

Implementation for Q-DEVS

This is the Adaptive Quantization implementation of the Watershed model.
The function used for local updating is the inverse function of the original (WSHED). This implementation is similar to the APAINV function of the Heart implementation showed before.

```
%Modelo Watershed implementado con Q-DEVS con funcion WSHINV
%-------------------------------------------------------------
%Ejecutar CON parametros -c00:00:01:000 (discrete time tik 1s)
%No cuantificable x simulacion. Cuantificable x parametro
%interno nro 6
%-------------------------------------------------------------

1   [top]
2   components : Watershed

3   [Watershed]
4   type : cell
5   dim : (16,28)
6   delay : transport
7   defaultDelayTime : 1000
8   border : nowrapped
9   neighbors :                     Watershed(-1,0)
10  neighbors :   Watershed(0,-1) Watershed(0,0) Watershed(0,1)
11  neighbors :                     Watershed(1,0)
12  initialvalue : 0.0
13  initialCellsValue : topo1528.val
14  localtransition : Hydrology

15  [Hydrology]
%0=>1, 1=>2, -1=>3

16  rule : { getArg(cellpos(0)*100000+cellpos(1)*100+8) }
    {
    setArg(cellpos(0)*100000+cellpos(1)*100+0,( 0, 0))+
    setArg(cellpos(0)*100000+cellpos(1)*100+1, if( (-1, 0) != ?,(-1, 0),-1.0) )+
    setArg(cellpos(0)*100000+cellpos(1)*100+2, if( ( 1, 0) != ?,( 1, 0),-1.0) )+
    setArg(cellpos(0)*100000+cellpos(1)*100+3, if( ( 0, 1) != ?,( 0, 1),-1.0)  )+
    setArg(cellpos(0)*100000+cellpos(1)*100+4, if( ( 0,-1) != ?,( 0,-1),-1.0)  )+
    setArg(cellpos(0)*100000+cellpos(1)*100+5,1000)+
    setArg(cellpos(0)*100000+cellpos(1)*100+6,1.0)+
    setArg(cellpos(0)*100000+cellpos(1)*100+7, WSHINV(cellpos(0),cellpos(1)))+
    getArg(cellpos(0)*100000+cellpos(1)*100+7)
    } { t }
```

The WSHINV function is called with the same arguments of the WSHED function with an additional one (argument number 6), which is the quantum value, used internally by the function (1.0 on the example).
The rule is first setting all the arguments with the setArg function and calling the WSHINV function, which returns and sets a new argument with the delay for this calculation. Finally, the getArg function is used to retrieve the new local value for the cell (getArg(x..y..8) returns the 8th argument for the x:y cell of the WSHINV function).
An example of the cells values (capability plus quantity of water) generated by a simulation with quantum 1.0 is showed above.

```
Line : 1203 - Time: 00:00:00:000
               0           1           2           3
    +-----------------------------------------------
   0|   100.00000    98.00000    96.00000    89.00000
   1|   100.00000    98.00000    96.00000    89.00000
   2|   100.00000    98.00000    96.00000    89.00000
   3|   100.00000   100.00000    98.00000    96.00000
   4|   100.00000   100.00000    98.00000    96.00000
   5|   100.00000   100.00000    98.00000    96.00000
   6|   100.00000   100.00000   100.00000    98.00000
   7|   100.00000   100.00000   100.00000    98.00000
   8|   100.00000   100.00000   100.00000   100.00000
   9|   100.00000   100.00000   100.00000   100.00000
  10|   100.00000   100.00000   100.00000   100.00000
  11|   100.00000   100.00000   100.00000   100.00000
  12|   100.00000   100.00000   100.00000    98.00000
  13|   100.00000   100.00000    98.00000    96.00000
  14|   100.00000    98.00000    96.00000    89.00000
  15|    98.00000    96.00000    89.00000    88.00000
```

```
    +-------------------------------------------------
                          ...
Line : 2812 - Time: 00:07:35:000
    +-------------------------------------------------
   0|   101.00024    99.00024    96.00000    90.00047
   1|   101.00009    99.00009    96.00000    90.00032
   2|   101.00009    99.00024    96.00000    90.00047
   3|   101.00055   100.00000    98.00000    96.00000
   4|   101.00055   100.00000    99.00009    96.00000
   5|   101.00055   100.00000    99.00024    96.00000
   6|   101.00055   101.00039   100.00000    98.00000
   7|   101.00055   101.00039   100.00000    99.00024
   8|   101.00055   101.00039   101.00039   100.00000
   9|   101.00055   101.00039   101.00039   101.00039
  10|   101.00055   101.00039   101.00039   101.00039
  11|   101.00055   101.00039   101.00039   100.00000
  12|   101.00055   101.00039   100.00000    98.00000
  13|   101.00055   100.00000    98.00000    96.00000
  14|   100.00000    98.00000    96.00000    90.00024
  15|    99.00039    96.00000    90.00062    89.00039
    +-------------------------------------------------
                          ...
Line : 4958 - Time: 00:10:00:000
    +-------------------------------------------------
   0|   101.00024    99.00024    97.00130    91.00103
   1|   101.00009    99.00009    97.00115    91.00068
   2|   101.00009    99.00024    97.00130    92.00637
   3|   101.00055   101.00168    99.00183    97.00174
   4|   101.00055   101.00213    99.00009    97.00115
   5|   101.00055   101.00213    99.00024    97.00130
   6|   101.00055   101.00039   101.00168    99.00183
   7|   101.00055   101.00039   101.00213    99.00024
   8|   101.00055   101.00039   101.00039   101.00168
   9|   101.00055   101.00039   101.00039   101.00039
  10|   101.00055   101.00039   101.00039   101.00039
  11|   101.00055   101.00039   101.00039   101.00168
  12|   101.00055   101.00039   101.00168    99.00198
  13|   101.00055   101.00168    99.00198    97.00294
  14|   101.00183    99.00198    97.00294    92.00658
  15|    99.00039    97.00145    92.00707    89.00039
    +-------------------------------------------------
```

Implementation for GDEVS

Here we have the GDEVS implementation of the Watershed model, using a similar mechanism of the GDEVS implementation showed in the Heart model. The difference here is that we have only one polynomial function with only one state. The function was approximated using a single polynomial function of order 1.
The a1 and b1 values for the a1*t + b1 polynomial function are 1/90 for a1 and 0 for b1. The delay of the rule is 20 seconds corresponding with the specified polynomial.
The polinomy used is defined by:

$$P_i(t) = a_i\, t + b_i \qquad \forall i \in [1, 1]$$

using the following coefficients:

i	ai	Bi	Time (ms)
1	0.01111…	0	[0, inf)

```
%Modelo Watershed implementado con GDEVS
%---------------------------------------------------------------
%Ejecutar CON parametros -c00:00:01:000 (discrete time tik 1s)
%---------------------------------------------------------------

1   [top]
2   components : Watershed

3   [Watershed]
4   type : cell
5   dim : (16,28)
```

```
6   delay : transport
7   defaultDelayTime : 1000
8   border : nowrapped
9   neighbors :                      Watershed(-1,0)
10  neighbors :   Watershed(0,-1) Watershed(0,0) Watershed(0,1)
11  neighbors :                      Watershed(1,0)
12  initialvalue : 0.0
13  initialCellsValue : topo1528.val
14  localtransition : Hydrology

15  [Hydrology]
%0=>1, 1=>2, -1=>3

16  rule :
    { (0,0) + ( 100 - getArg(cellpos(0)*100000+cellpos(1)*100 + 0) ) / 90 }
    { 20000 + if( getArg(cellpos(0)*100000+cellpos(1)*100 + 0)=0,
    setArg(cellpos(0)*100000+cellpos(1)*100 + 0, (0,0)), 0) }
    { (0,0) < 100.0 }

17  rule : { (0,0) } 20000 { t }
```

This model is like the Cell-DEVS one, but the difference is on the rule, which has the polynomial function instead of the WSHED function.

With the polynomial definitions we can define the actions of each of the cell's local computing functions, which are described by the state graph in the following Figure19. The Figure defines a G-DEVS model with the classical state transition functions using event of order 1.

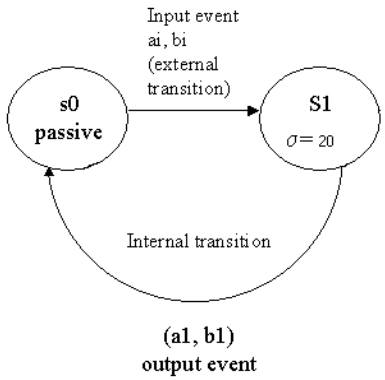


Figure 19. GDEVS specification of a cell.

An example of the cells capability plus accumulated water values generated by a simulation is showed above.

```
Line : 1353 - Time: 00:00:00:000
               0          1          2          3
    +-----------------------------------------------
   0|   100.00000    98.00000    96.00000    89.00000
   1|   100.00000    98.00000    96.00000    89.00000
   2|   100.00000    98.00000    96.00000    89.00000
   3|   100.00000   100.00000    98.00000    96.00000
   4|   100.00000   100.00000    98.00000    96.00000
   5|   100.00000   100.00000    98.00000    96.00000
   6|   100.00000   100.00000   100.00000    98.00000
   7|   100.00000   100.00000   100.00000    98.00000
   8|   100.00000   100.00000   100.00000   100.00000
   9|   100.00000   100.00000   100.00000   100.00000
  10|   100.00000   100.00000   100.00000   100.00000
  11|   100.00000   100.00000   100.00000   100.00000
  12|   100.00000   100.00000   100.00000    98.00000
  13|   100.00000   100.00000    98.00000    96.00000
  14|   100.00000    98.00000    96.00000    89.00000
  15|    98.00000    96.00000    89.00000    88.00000
    +-----------------------------------------------
```

```
Line : 3537 - Time: 00:00:20:000
    +-----------------------------------------------
   0|   100.00000    98.02222    96.04444    89.12222
   1|   100.00000    98.02222    96.04444    89.12222
   2|   100.00000    98.02222    96.04444    89.12222
   3|   100.00000   100.00000    98.02222    96.04444
   4|   100.00000   100.00000    98.02222    96.04444
   5|   100.00000   100.00000    98.02222    96.04444
   6|   100.00000   100.00000   100.00000    98.02222
   7|   100.00000   100.00000   100.00000    98.02222
   8|   100.00000   100.00000   100.00000   100.00000
   9|   100.00000   100.00000   100.00000   100.00000
  10|   100.00000   100.00000   100.00000   100.00000
  11|   100.00000   100.00000   100.00000   100.00000
  12|   100.00000   100.00000   100.00000    98.02222
  13|   100.00000   100.00000    98.02222    96.04444
  14|   100.00000    98.02222    96.04444    89.12222
  15|    98.02222    96.04444    89.12222    88.13333
    +-----------------------------------------------
                        ...
Line : 34099 - Time: 00:05:00:000
    +-----------------------------------------------
   0|   100.00000    98.30862    96.61723    90.69738
   1|   100.00000    98.30862    96.61723    90.69738
   2|   100.00000    98.30862    96.61723    90.69738
   3|   100.00000   100.00000    98.30862    96.61723
   4|   100.00000   100.00000    98.30862    96.61723
   5|   100.00000   100.00000    98.30862    96.61723
   6|   100.00000   100.00000   100.00000    98.30862
   7|   100.00000   100.00000   100.00000    98.30862
   8|   100.00000   100.00000   100.00000   100.00000
   9|   100.00000   100.00000   100.00000   100.00000
  10|   100.00000   100.00000   100.00000   100.00000
  11|   100.00000   100.00000   100.00000   100.00000
  12|   100.00000   100.00000   100.00000    98.30862
  13|   100.00000   100.00000    98.30862    96.61723
  14|   100.00000    98.30862    96.61723    90.69738
  15|    98.30862    96.61723    90.69738    89.85169
    +-----------------------------------------------
                        ...
Line : 66392 - Time: 00:10:00:000
    +-----------------------------------------------
   0|   100.00000    98.56961    97.13922    92.13285
   1|   100.00000    98.56961    97.13922    92.13285
   2|   100.00000    98.56961    97.13922    92.13285
   3|   100.00000   100.00000    98.56961    97.13922
   4|   100.00000   100.00000    98.56961    97.13922
   5|   100.00000   100.00000    98.56961    97.13922
   6|   100.00000   100.00000   100.00000    98.56961
   7|   100.00000   100.00000   100.00000    98.56961
   8|   100.00000   100.00000   100.00000   100.00000
   9|   100.00000   100.00000   100.00000   100.00000
  10|   100.00000   100.00000   100.00000   100.00000
  11|   100.00000   100.00000   100.00000   100.00000
  12|   100.00000   100.00000   100.00000    98.56961
  13|   100.00000   100.00000    98.56961    97.13922
  14|   100.00000    98.56961    97.13922    92.13285
  15|    98.56961    97.13922    92.13285    91.41765
    +-----------------------------------------------
```

6 Flow Injection Analysis Model Description and Implementations

[17] Flow-injection methods are analytical methods used for automated sample analysis of liquid samples. In a flow injection analyzer, a small and fixed volume of a liquid sample is injected as a discrete zone using an injection device into a liquid carrier, which flows through a narrow tube. As a result of convection at the beginning, and later of axial and radial diffusion, this sample is progressively dispersed into the carrier as it is transported along the tube. The addition of reagents at different confluence points (which mix with the sample as a result of radial dispersion) produces reactive or detectable species, which can be sensed by flow-through detection devices. Figure 20 presents a simple flow-injection apparatus.

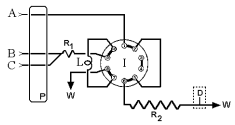


Figure 20 : A FIA manifold.

This device (called a FIA manifold) consists on a peristaltic pump (P) that adds carrier solution (A) into a valve (I) that connects to a tube called a reactor (R2). At the end of the tube a detector is placed to sense a specific property of the flowing solution. The valve can be turned to allow the flow of the sample (B) into the reactor. The sample is held in the loop L and when the valve is rotated its contents flow into the reactor, where chemical activity will usually take place between the sample and the carrier solution. As a result, a change will be observed in the signal produced by D, making it possible to quantify the sample after comparing the results with those obtained by known samples.
In a FI system convective transport yields a parabolic velocity profile with molecules at the tube walls having speed zero and those at the center having twice the average velocity. At the same time, the presence of concentration gradients develops axial and radial diffusion of sample molecules. It has been reported that in FI systems of practical interest, axial molecular diffusion has almost no influence in the overall dispersion, but radial diffusion is the main contributor. For a pump proving a net flow of q ml/min in a coil of radius a, the average flow velocity is given by:

$$V_a = \frac{q}{60 \cdot (\pi \cdot a^2)} \qquad \textbf{(Equation 1)}$$

At a point at distance r from the center, the flow velocity is described by:

$$v(r) = 2 \cdot V_a \cdot \left(1 - \frac{r^2}{a^2}\right) \qquad \textbf{(Equation 2)}$$

It is very difficult, if not impossible, to correlate the experimentally obtained response curve with the actual spatial mass distribution of the system. This is a consequence of the selected method of measurement, which fixes spatially and temporally the point of detection. Under these circumstances, any event occurred before the detection point is inferred from the response curve profile. Therefore, this detection approach is a powerful tool for predicting response curves, but ignores the processes leading to the generation of such response. A FI system using nitric acid as the carrier solution, water as the injected sample and a digital conductimeter with a couple of wires at both ends of the carrier stream detector was used to follow the radial mass distribution of the sample zone.

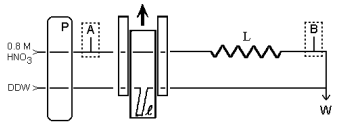


Figure 21 : FIA manifold for continuously monitoring.
P = pump; l = loop; L = reactor; W=waste; A, B = detection points. Punctual detection: suitable detector in point B; integrated detection: Pt wires located at points A-B.

When the water sample is injected, it acts as a blocking disc, and no electric conductance is measured. As convective transport and diffusion gradient forces the water sample to be released from the walls, causing a reduction of the blocking area and allowing electric current to flow, conductivity values different from zero are measured. Figure 22 shows the characteristic conductivity curve obtained by such a system.

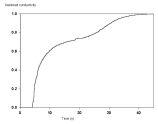


Figure 22 : Characteristic conductivity curve [1]

Original CD++ Implementation

[17] As mentioned, it is impossible to analyze the detailed behavior of the changes in the mass distribution profile. Therefore, we decided to build a Cell-DEVS model describing the integrated conductivity flow-injection system (ICM) in detail. In this way, the internal complex behavior can be

analyzed by studying the simulated results. The ICM system consists of a 0.025 cm radius tube, a 10.75 cm loop and a 9,25 reactor coil . We assumed the total tube length of the tube to be of 20cm. For this system, a cell space of 25 rows and 200 columns was defined, each cell representing a 0.001 x 0.1cm of a half tube section. Row 0 represents the center of the tube and row 24 the section of the tube touching its walls and the value of each cell will represent the nitric acid concentration.

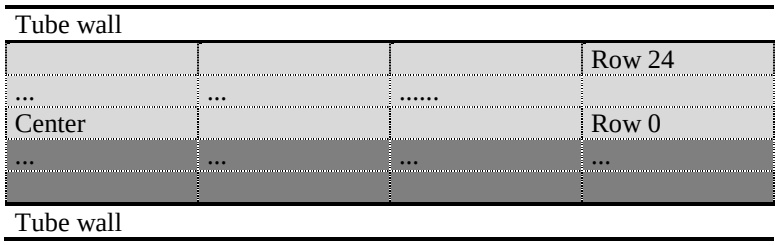


Figure 23 : Correspondence between the cell-space and the actual tube

Figure 23 shows in light gray a tube section representing a cell. This is a longitudinal cut of the tube. The final aim is to build a 3 dimensional space representing a cylindrical section of the tube, but in this case each cell represent a flat section.

To deal with convective transport and radial diffusion at the same time, the model reacts in two phases: transport and diffusion. The local computing function simulates the transport phase, and all cells are connected to an external generator sending an event, which triggers the diffusion phase. The model is built as a coupled DEVS model with two components: a Cell-DEVS (named fia) representing the tube, and an atomic model (named generator). The generator has one output port (out) to send the diffusion-triggering event. This port is mapped to the diffuse input port of the fia model (line 2). This means all output events sent through the out port will be received as external events by the fia model through the diffuse port.

```
00 [Top]
01 components :  fia generator@ConstGenerator
02 link : out@generator diffuse@fia
03
04 [generator]
05 frecuency : 00:00:00:014
```

Figure 24 : Components of the DEVS model

Equation 3 defines the frequency of diffuse events. This equation computes the characteristic distance a particle of a given solution of diffusion coefficient c will travel in dt seconds.

$$ds = \sqrt{2 \cdot c \cdot dt} \qquad \textbf{(Equation 3)}$$

Solving the equation for c = 3,5 x 10^{-5} cm/s and ds = 0.001 cm, we obtain a dt of 14ms. We used for the ds value the cell height to find out how long it would take for two cells to diffuse homogeneously. We did not take into account the cell width because axial diffusion can be ignored.

```
05 [fia]
06 in : diffuse
07 width : 200
08 height : 25
09 delay :   inertial
10 border :   nowrapped
11 neighbors : fia(-1,-1) fia(-1,0) fia(-1,1)
12 neighbors : fia(0,-1) fia(0,0) fia(0,1)
13 neighbors : fia(1,-1) fia(1,0) fia(1,1)
14 localtransition :  transport
```

Figure 25 : Definition of the FIA coupled cell model

Figure 25 shows the definition of the parameters for the coupled Cell-DEVS fia. Line 6 defines the diffuse input port, and lines 7 and 8 define the cell space dimensions. Line 9 sets the cell delay type to

inertial. An inertial delay cell that has a scheduled future value f will preempt this value if upon receiving an external event and evaluating the local transition rules a new future value f_1,with $f \neq f_1$, is obtained. In this case, f_1 will be scheduled as the future value with a given delay d. Line 10 defines non-wrapped borders and lines 11 to 13 define a cell's neighborhood shape. Finally, line 14 defines the sets the local transition function rules, which is defined in Figure 26.

```
18 [transport]
19 rule : { (0,-1) } { 0.1 / ( 22.57878 * ( 1 - power( cellPos(0) * 0.001 + 0.0005 , 2)
        / 0.000625 )) * 1000 } { cellpos(1) != 0 }
20 rule : { 0.8 } { 0.1 / ( 22.57878 * ( 1 - power( cellPos(0) * 0.001 + 0.0005 , 2) /
        0.000625 )) * 1000 } { cellpos(1) = 0 }
```

Figure 26 : The local transition rules

The convective transport has been arbitrarily been defined in the direction of increasing column values, so that in visual representations the carrier will be seen flowing from left to right. Being this the case, a local transition rule for the transport phase should set a cell's value to the current value of its (0,-1) neighbor cell. The rate at which this is done depends on the velocity of the flow at the cell, which, as mentioned before, has its maximum at the center of the tube and decreases towards its walls. This is stated in the first transport rule in line 19. The three components of the local transition rule are:

Value: `{ (0,-1) }  //The value of the cell's left neighbor`

Delay:
```
{ 0.1 / ( 22.57878 * ( 1 - power( cellPos(0) * 0.001 + 0.0005 , 2)
/ 0.000625 )) * 1000 }
```

Condition: `{ cellpos(1) != 0 }`

Figure 27 shows the radial diffusion rules. For a cell with valid top and bottom neighbors, the diffusion rule states that the new cell value will be the average of the three cells. This is the case of the rule in line 22. A delay of 1 ms was chosen. Though a 0 ms delay would be more appropriate, this is still not supported in the version of NCD++ for which the model was written. The other three rules in lines 23 and 24 cover the special case of top and border cells. These cells do not have both, a valid top and bottom neighbor so instead of using three cells to obtain the average, only two are used.

```
21 [diffusion]
22 rule : { ((-1,0) + (0,0) + (1,0)) / 3 } 1 { cellpos(0) != 0 AND cellpos(0) != 24 }
23 rule : { ((-1,0) + (0,0)) / 2 } 1 { cellpos(0) != 0 AND cellpos(0) = 24 }
24 rule : { ((0,0) +  (1,0)) / 2 } 1 { cellpos(0) = 0 AND cellpos(0) != 24 }
```

Figure 27 : Radial diffusion rules.

So far we have shown the diffusion rule, but we have not yet defined that this ruled should be evaluated when an external event is received through the diffuse input port. Figure 28 shows the statements that link the fia model diffuse input port to a cell's diffuse input port (line 27) and set the diffusion rule to be evaluated upon the arrival of an external event through this port (line 28).

```
[fia]
27 link : diffuse diffuse@fia(x,y)
28 PortInTransition : diffuse@fia(x,y) diffusion
```

Figure 28 : External coupling of the FIA Cell-DEVS model.

The described model was run for 10s and the state of the whole cell space was logged every 100ms. A graphical representation of the model at five different stages is shown in Figure 29. The logged results were also used to draw the conductivity curve.

To obtain the conductivity of the whole system, we divided the cell space in axial segments, calculated the resistance of each, and assumed the whole resistance to be the result of combining all segments in serial mode. We took each segment to be a column of cells and calculated its resistance.

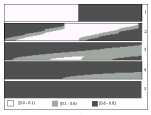


Figure 29 : Different execution stages of the FIA model.

On figure 29: (1) At time 0 the sample (white), has been injected. The other half of the tube contains the carrier solution (dark gray). (2,3,4) The convective transport makes the sample disperse faster at the middle of the tube than near the walls. (5) The whole tube now contains the carrier solution only.

New CD++ Implementation

The following is the FIA model simplified as we did it with previous models, introducing the FIA function into CD++. As we can see, on this new implementation, we have not the attached models and the rules were simplified with the use of the internal FIA function.

```
1)  [Top]
2)  components :  fia

3)  [fia]
4)  type : cell
5)  dim : (20,80)
6)  delay :   inertial
7)  defaultDelayTime  : 1
8)  border :   nowrapped
9)  neighbors : fia(-1,-1) fia(-1,0) fia(-1,1)
10) neighbors : fia(0,-1) fia(0,0) fia(0,1)
11) neighbors : fia(1,-1) fia(1,0) fia(1,1)
12) initialvalue : 0.8
13) initialcellsvalue : newfia5x20.val
14) localtransition :  fia-rule


15) [fia-rule]

16) rule : { getArg(cellpos(0)*100000+cellpos(1)*100+8) }
    {
     setArg(cellpos(0)*100000+cellpos(1)*100+0,(0,0))+
     setArg(cellpos(0)*100000+cellpos(1)*100+1,(1,0))+
     setArg(cellpos(0)*100000+cellpos(1)*100+2,(-1,0))+
     setArg(cellpos(0)*100000+cellpos(1)*100+3,(0,-1))+
     setArg(cellpos(0)*100000+cellpos(1)*100+4,time)+
     setArg(cellpos(0)*100000+cellpos(1)*100+5,11)+
     setArg(cellpos(0)*100000+cellpos(1)*100+6,80)+
     setArg(cellpos(0)*100000+cellpos(1)*100+7,20) +
     setArg(cellpos(0)*100000+cellpos(1)*100+10,0) +
     setArg(cellpos(0)*100000+cellpos(1)*100+11,0) +
     setArg(cellpos(0)*100000+cellpos(1)*100+8, FIA(cellpos(0), cellpos(1)))+
     getArg(cellpos(0)*100000+cellpos(1)*100+9)
    }
    { t }
```

Line **1 and 2** defines the Cell-DEVS model fia.
Line **3** describes the model. Line **4** indicates CD++ that is a cellular model.
Line **5** defines the dimension of the cellular space. On this case we have a 2d cellular spaces of 20x80 cells. Line **6** defined the delay type of each cell, which is inertial on this case.
Line **7** defined the default delay time on 1 millisecond.
Line **8** defines the border type no wrapped, so the function FIA has an special behavior on borders (that's why FIA function receives the dimension as arguments 6 and 7).
Lines **9** through **11** define the relative positions of the neighbors.
Line **12** defines de default value of each cell to start the simulation, which is 0.8.
Line **13** indicates CD++ the file with the initial values. In this case, some cells are specified with a 0 value which represent the injected sample on the tube for the FIA analysis.
Line **14** defines the name of the rules behavior section.
On line **16** we have the first (and unique) rule of the model, using the FIA function, which receives 10 arguments:

- The current value of the analyzed cell
- The current value of three of the neighbors, which are used on the computation of the new concentration of sample on each cell.
- The simulation time to determine if diffusion or a transport activity action is required.
- The diffusion interval in use.
- Then we have two arguments for the dimension of the cell space.
- An indicator (not used on this work) to force diffusion at time 0 (when the simulation starts).

- An indicator to use a different diffusion technique (the average of the neighbors) not used on this work.

An example of the cells liquid concentration values generated by a simulation without quantum is showed above.

```
Line : 10252 - Time: 00:00:00:000
              0          1          2          3          4
    +-------------------------------------------------------------
   0|     0.00000     0.00000     0.00000     0.00000     0.00000
   1|     0.00000     0.00000     0.00000     0.00000     0.00000
   2|     0.00000     0.00000     0.00000     0.00000     0.00000
   3|     0.00000     0.00000     0.00000     0.00000     0.00000
   4|     0.00000     0.00000     0.00000     0.00000     0.00000
   5|     0.00000     0.00000     0.00000     0.00000     0.00000
   6|     0.00000     0.00000     0.00000     0.00000     0.00000
   7|     0.00000     0.00000     0.00000     0.00000     0.00000
   8|     0.00000     0.00000     0.00000     0.00000     0.00000
   9|     0.00000     0.00000     0.00000     0.00000     0.00000
  10|     0.00000     0.00000     0.00000     0.00000     0.00000
  11|     0.00000     0.00000     0.00000     0.00000     0.00000
  12|     0.00000     0.00000     0.00000     0.00000     0.00000
  13|     0.00000     0.00000     0.00000     0.00000     0.00000
  14|     0.00000     0.00000     0.00000     0.00000     0.00000
  15|     0.00000     0.00000     0.00000     0.00000     0.00000
  16|     0.00000     0.00000     0.00000     0.00000     0.00000
  17|     0.00000     0.00000     0.00000     0.00000     0.00000
  18|     0.00000     0.00000     0.00000     0.00000     0.00000
  19|     0.00000     0.00000     0.00000     0.00000     0.00000
    +-------------------------------------------------------------
                        ...
Line : 76641 - Time: 00:00:00:049
    +-------------------------------------------------------------
   0|     0.80000     0.80000     0.80000     0.80000     0.80000
   1|     0.80000     0.80000     0.80000     0.80000     0.80000
   2|     0.80000     0.80000     0.80000     0.80000     0.80000
   3|     0.80000     0.80000     0.80000     0.80000     0.80000
   4|     0.80000     0.80000     0.80000     0.80000     0.80000
   5|     0.80000     0.80000     0.80000     0.80000     0.80000
   6|     0.80000     0.80000     0.80000     0.80000     0.80000
   7|     0.80000     0.80000     0.80000     0.80000     0.80000
   8|     0.80000     0.80000     0.80000     0.80000     0.60015
   9|     0.80000     0.80000     0.80000     0.80000     0.71787
  10|     0.80000     0.80000     0.80000     0.80000     0.75015
  11|     0.80000     0.80000     0.78841     0.73859     0.69308
  12|     0.78933     0.64040     0.59852     0.54123     0.45985
  13|     0.79012     0.36580     0.32702     0.27398     0.20850
  14|     0.80000     0.39574     0.36884     0.26394     0.14100
  15|     0.80000     0.55054     0.55054     0.27240     0.18925
  16|     0.51717     0.70572     0.60606     0.25589     0.17778
  17|     0.00000     0.71111     0.33524     0.08381     0.08381
  18|     0.80000     0.80000     0.00000     0.00000     0.00000
  19|     0.00000     0.00000     0.00000     0.00000     0.00000
    +-------------------------------------------------------------
                        ...
Line : 8357284 - Time: 00:00:05:000
    +-------------------------------------------------------------
   0|     0.80000     0.80000     0.80000     0.80000     0.80000
   1|     0.80000     0.80000     0.80000     0.80000     0.80000
   2|     0.80000     0.80000     0.80000     0.80000     0.80000
   3|     0.80000     0.80000     0.80000     0.80000     0.80000
   4|     0.80000     0.80000     0.80000     0.80000     0.80000
   5|     0.80000     0.80000     0.80000     0.80000     0.80000
   6|     0.80000     0.80000     0.80000     0.80000     0.80000
   7|     0.80000     0.80000     0.80000     0.80000     0.80000
   8|     0.80000     0.80000     0.80000     0.80000     0.80000
   9|     0.80000     0.80000     0.80000     0.80000     0.80000
  10|     0.80000     0.80000     0.80000     0.80000     0.80000
  11|     0.80000     0.80000     0.80000     0.80000     0.80000
  12|     0.80000     0.80000     0.80000     0.80000     0.80000
  13|     0.80000     0.80000     0.80000     0.80000     0.80000
  14|     0.80000     0.80000     0.80000     0.80000     0.80000
  15|     0.80000     0.80000     0.80000     0.69964     0.80000
  16|     0.51717     0.80000     0.65051     0.50909     0.65051
  17|     0.00000     0.80000     0.37714     0.24381     0.37714
  18|     0.80000     0.80000     0.00000     0.00000     0.00000
```

```
 19|     0.00000     0.00000     0.00000     0.00000     0.00000
   +-------------------------------------------------------------
```

Implementation for Q-DEVS

This is the Adaptive Quantization implementation for the FIA model.
The function used for local updating is the inverse function of the original (FIA). This implementation is similar to the Heart and Watershed implementation showed before.

```
%Modelo FIA implementado con Q-DEVS
%-------------------------------------------------------------
%NO cuantificable x simulacion. Cuantificable internamente
%con parametro 12 de la funcion FIAINV
%-------------------------------------------------------------
1   [Top]
2   components :  fia

3   [fia]
4   type : cell
5   dim : (20,80)
6   %width : 20
7   %height : 5
8   delay :   inertial
9   defaultDelayTime  : 1
10  border :   nowrapped
11  neighbors : fia(-1,-1) fia(-1,0) fia(-1,1)
12  neighbors : fia(0,-1) fia(0,0) fia(0,1)
13  neighbors : fia(1,-1) fia(1,0) fia(1,1)
14  initialvalue : 0.8
15  initialcellsvalue : newfia5x20.val
16  localtransition :  fia-rule


17  [fia-rule]
18  rule : { getArg(cellpos(0)*100000+cellpos(1)*100+13) }
    {
    setArg(cellpos(0)*100000+cellpos(1)*100+0,(0,0))+
    setArg(cellpos(0)*100000+cellpos(1)*100+1,(1,0))+
    setArg(cellpos(0)*100000+cellpos(1)*100+2,(-1,0))+
    setArg(cellpos(0)*100000+cellpos(1)*100+3,(0,-1))+
    setArg(cellpos(0)*100000+cellpos(1)*100+4,time)+
    setArg(cellpos(0)*100000+cellpos(1)*100+5,11)+
    setArg(cellpos(0)*100000+cellpos(1)*100+6,80)+
    setArg(cellpos(0)*100000+cellpos(1)*100+7,20) +
    setArg(cellpos(0)*100000+cellpos(1)*100+10,0) +
    setArg(cellpos(0)*100000+cellpos(1)*100+11,0) +
    setArg(cellpos(0)*100000+cellpos(1)*100+12,0.7) +
    FIAINV(cellpos(0), cellpos(1))
    }
    { t }
```

The FIAINV function is called with the same arguments of the FIA function plus an additional argument, which is the quantum value, used internally by the function (0.7 on this example).
The rule is first setting all the arguments with the setArg function and calling the FIAINV function, which returns the delay for this calculation. Finally the getArg function is called to retrieve the new local value for the cell (getArg(x..y..13) which retrieves the 13th argument value (which is the calculated value of sample concentration) for the cell x:y).
An example of the cells sample concentration values generated by a simulation with quantum 0.7 is showed above.

```
Line : 10252 - Time: 00:00:00:000
                0           1           2           3           4
    +-------------------------------------------------------------
   0|     0.00000     0.00000     0.00000     0.00000     0.00000
   1|     0.00000     0.00000     0.00000     0.00000     0.00000
   2|     0.00000     0.00000     0.00000     0.00000     0.00000
   3|     0.00000     0.00000     0.00000     0.00000     0.00000
   4|     0.00000     0.00000     0.00000     0.00000     0.00000
   5|     0.00000     0.00000     0.00000     0.00000     0.00000
   6|     0.00000     0.00000     0.00000     0.00000     0.00000
   7|     0.00000     0.00000     0.00000     0.00000     0.00000
   8|     0.00000     0.00000     0.00000     0.00000     0.00000
```

```
   9|     0.00000     0.00000     0.00000     0.00000     0.00000
  10|     0.00000     0.00000     0.00000     0.00000     0.00000
  11|     0.00000     0.00000     0.00000     0.00000     0.00000
  12|     0.00000     0.00000     0.00000     0.00000     0.00000
  13|     0.00000     0.00000     0.00000     0.00000     0.00000
  14|     0.00000     0.00000     0.00000     0.00000     0.00000
  15|     0.00000     0.00000     0.00000     0.00000     0.00000
  16|     0.00000     0.00000     0.00000     0.00000     0.00000
  17|     0.00000     0.00000     0.00000     0.00000     0.00000
  18|     0.00000     0.00000     0.00000     0.00000     0.00000
  19|     0.00000     0.00000     0.00000     0.00000     0.00000
    +-------------------------------------------------------------
                              ...
Line : 75746 - Time: 00:00:00:049
    +-------------------------------------------------------------
   0|     0.80000     0.80000     0.80000     0.80000     0.80000
   1|     0.80000     0.80000     0.80000     0.80000     0.80000
   2|     0.80000     0.80000     0.80000     0.80000     0.80000
   3|     0.80000     0.80000     0.80000     0.80000     0.80000
   4|     0.80000     0.80000     0.80000     0.80000     0.80000
   5|     0.80000     0.80000     0.80000     0.80000     0.80000
   6|     0.80000     0.80000     0.80000     0.80000     0.80000
   7|     0.80000     0.80000     0.80000     0.49778     0.49778
   8|     0.00000     0.00000     0.00000     0.00000     0.00000
   9|     0.56140     0.56140     0.00000     0.00000     0.00000
  10|     0.80000     0.80000     0.55873     0.55873     0.55873
  11|     0.80000     0.80000     0.51014     0.51014     0.00000
  12|     0.51200     0.51200     0.00000     0.00000     0.00000
  13|     0.00000     0.00000     0.00000     0.00000     0.00000
  14|     0.80000     0.80000     0.80000     0.80000     0.80000
  15|     0.00000     0.00000     0.00000     0.00000     0.00000
  16|     0.26667     0.26667     0.26667     0.26667     0.00000
  17|     0.00000     0.00000     0.00000     0.00000     0.00000
  18|     0.80000     0.80000     0.80000     0.80000     0.80000
  19|     0.00000     0.00000     0.00000     0.00000     0.00000
    +-------------------------------------------------------------
                              ...
Line : 6182722 - Time: 00:00:05:000
    +-------------------------------------------------------------
   0|     0.80000     0.80000     0.80000     0.80000     0.80000
   1|     0.80000     0.80000     0.80000     0.80000     0.80000
   2|     0.80000     0.80000     0.80000     0.80000     0.80000
   3|     0.80000     0.80000     0.80000     0.80000     0.80000
   4|     0.80000     0.80000     0.80000     0.80000     0.80000
   5|     0.80000     0.80000     0.80000     0.80000     0.80000
   6|     0.80000     0.80000     0.80000     0.80000     0.80000
   7|     0.80000     0.80000     0.80000     0.80000     0.49778
   8|     0.00000     0.00000     0.00000     0.00000     0.00000
   9|     0.80000     0.56140     0.00000     0.00000     0.00000
  10|     0.80000     0.80000     0.80000     0.55873     0.55873
  11|     0.80000     0.80000     0.80000     0.51014     0.00000
  12|     0.80000     0.51200     0.00000     0.00000     0.00000
  13|     0.00000     0.00000     0.00000     0.00000     0.00000
  14|     0.80000     0.80000     0.80000     0.80000     0.80000
  15|     0.00000     0.00000     0.00000     0.00000     0.00000
  16|     0.26667     0.26667     0.26667     0.26667     0.26667
  17|     0.00000     0.00000     0.00000     0.00000     0.00000
  18|     0.80000     0.80000     0.80000     0.80000     0.80000
  19|     0.00000     0.00000     0.00000     0.00000     0.00000
    +-------------------------------------------------------------
```

7 Simulation Analysis

This section shows the results of the different simulation tests made on each of the three analyzed models. The error, time and messages are analyzed with different quantums types, techniques and values that are also described on next sections.

7.1 Simulations combiations description

Large simulations experiments and the results of applying the different quantum techniques and types are showed.

All the different combinations of quantum types and techniques were tested. Most representative quantum values where selected for each model. The selection was made after several tries with different values for each model. For dynamic strategies 1 and 2, most representative ratios where used as well.
The analyzed combinations were:

- Quantum Standard.
- Quantum Standard with dynamic quantum strategy 1.
- Quantum Standard with dynamic quantum strategy 2.
- Quantum Hysteresis.
- Quantum Hysteresis with dynamic quantum strategy 1.
- Quantum Hysteresis with dynamic quantum strategy 2.
- Adaptive Quantization (Q-DEVS)
- Generalized Discrete Event Simulation (GDEVS)
- Generalized Discrete Event Simulation (GDEVS) with Standard quantum.
- Generalized Discrete Event Simulation (GDEVS) with Standard and dynamic quantum strategy 1.
- Generalized Discrete Event Simulation (GDEVS) with Standard and dynamic quantum strategy 2.
- Generalized Discrete Event Simulation (GDEVS) with Hysteresis quantum.
- Generalized Discrete Event Simulation (GDEVS) with Hysteresis and dynamic quantum strategy 1.
- Generalized Discrete Event Simulation (GDEVS) with Hysteresis and dynamic quantum strategy 2.

In the next sections, the simulation results of each model with its quantum combinations are showed.

7.2 Heart Model

The Heart model was tested using the following selected quantums types and values, showed on the tree Figure 32 below:

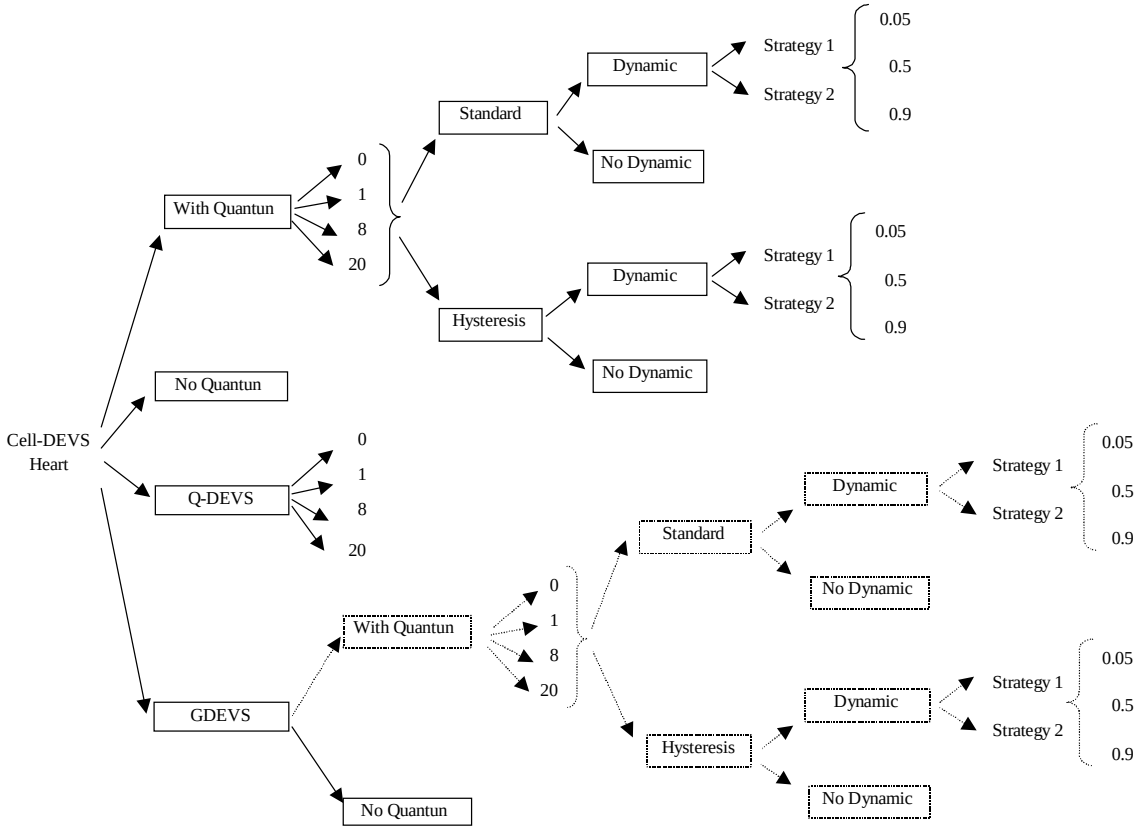
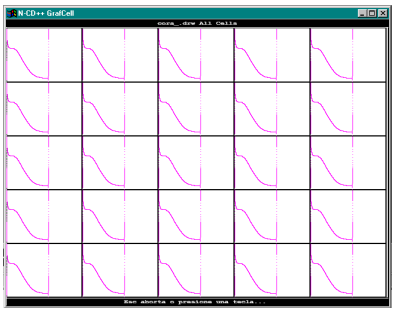


Figure 32. Quantum types and techniques used with Heart model

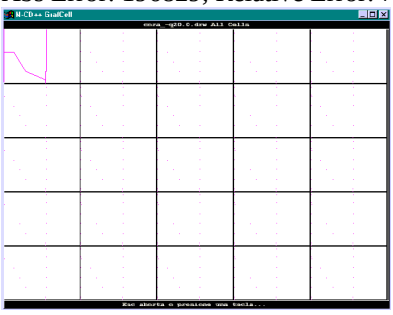
GDEVS with quantum is showed with "….." lines in order to indicate that there was no necessary to test that combination, but was done and some different results were obtained.
The values showed on the dynamic quantum are the different ratios used for that quantums types.
Figure 32 shows the values used with Cell-DEVS, GDEVS and Q-DEVS mechanisms.
Standard and Hysteresis quantum were analyzed with all the specified values and combination of dynamic strategy 1 and 2 and non-dynamic as well.

Next figure (Figure 33) shows simulation examples of the graphical output with the quantum type and values used. This graphics are a subset of all the simulations made and were drawn using the graphcell toll.

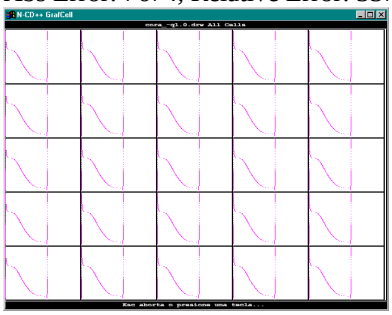
No Quantum
Messages: Y= 390.000, All= 2.012.454



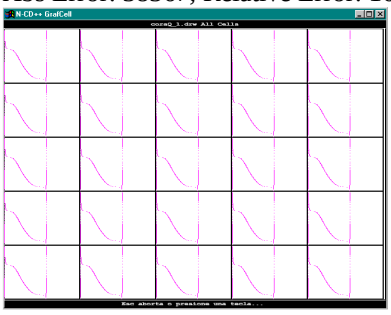
Quantum Standard 20
Messages: Y= 475, All= 1.622.929
Abs Error: 196829, Relative Error: 7879



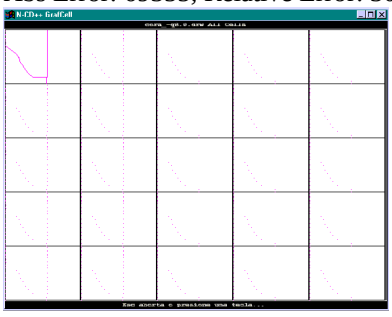
Quantum Standard 1
Messages: Y= 6075, All= 1.628.529
Abs Error: 7674, Relative Error: 337



Quantum Q-DEVS 1
Messages: Y= 6075, All= 31.401
Abs Error: 38367, Relative Error: 1686



Quantum Standard 8
Messages: Y= 1125, All= 1.623.579
Abs Error: 69355, Relative Error: 3020



GDEVS
Messages: Y= 2075, All= 10761
Abs Error: 1.388.980, Relative Error: 31912

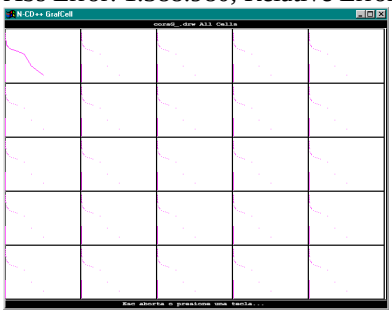


Figure 33. Heart model result simulation examples

On the above figures, different graphical output generations can be seen with different (less) graphical precision because of quantization. Lines with "…." are showed when data is not available because of quantums discretization (a few values every certain amount of time is showed). On the first cell of each graph the dots lines were over drawn (with tendency lines) to show the approximation made when

different quantum techniques are in use. For instance, on the quantum standard 20 figure, cell 0 0 was overdrawed with lines (the other cells not, so doted lines are showed).

Error analysis

On this section, the error is analyzed for each quantum value, type and combinations of them for Cell-DEVS, Q-DEVS and GDEVS formalisms.

The analysis is based on the error analysis, which was calculated comparing the values at each time with the Heart Cell-DEVS simulation without quantum (that is our base case for the result comparisons of the different techniques).

The absolute error was calculated as follows:

Absolute error at time t for the cell i = $(s_{i,t} - q_{i,t})$

Where

$s_{i,t}$ is the value of the Cell-DEVS simulation without quantum of the cell i at time t

$q_{i,t}$ is the value of the quantized Cell-DEVS simulation of the cell i at time t

Absolute error at time t for all the cells = $\sum_{i=1..n} (s_{i,t} - q_{i,t}) / n$

Where:

n is the total number of cells (n = x * y if the cellular space of the model is defined as x*y cells)

The accumulated absolute error of the complete simulation = $\sum_{t=0,\ t<\delta} \left(\sum_{i=1..n} (s_{i,t} - q_{i,t}) \right) / n$

Where:

i is the i^{th} cell.

δ is the programmed ending simulation time.

The relative error was calculated as follows:

Relative error at time t for the cell i = $(s_{i,t} - q_{i,t}) / s_{i,t}$

Where

$s_{i,t}$ is the value of the Cell-DEVS simulation without quantum of the cell i at time t

$q_{i,t}$ is the value of the quantized Cell-DEVS simulation of the cell i at time t

Relative error at time t for all the cells = $\sum_{i=1..n} ((s_{i,t} - q_{i,t}) / s_{i,t}) / n$

Where:

n is the total number of cells (n = x * y if the cellular space of the model is defined as x*y cells)

The accumulated relative error of the complete simulation = $\sum_{t=0,\ t<\delta} (\sum_{i=1..n} ((s_{i,t} - q_{i,t}) / s_{i,t})) / n$

Where:

i is the i^{th} cell.

δ is the programmed ending simulation time.

Absolute Error analysis

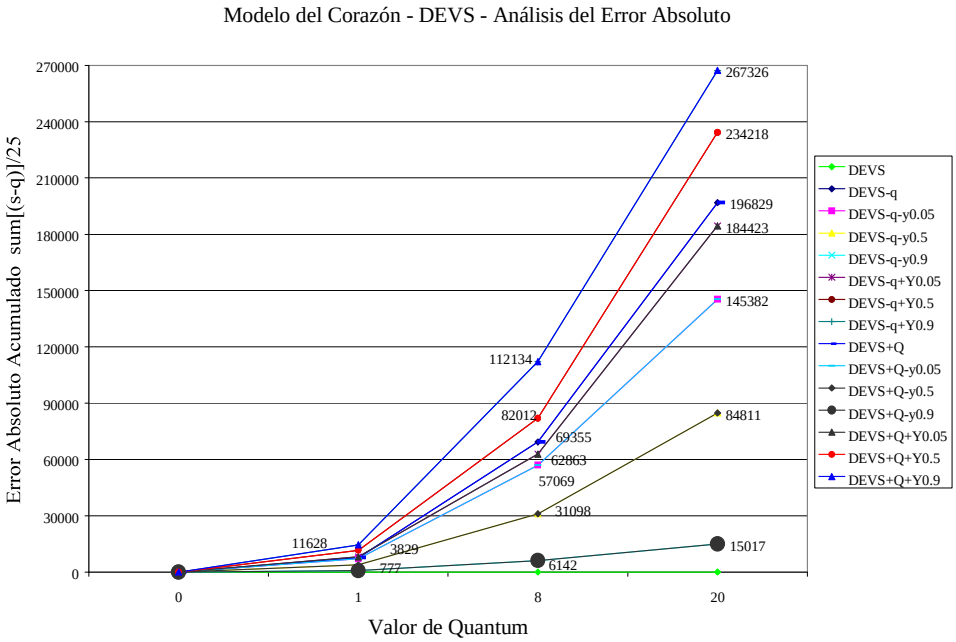


Figure 34. Cell-DEVS Heart model Absolute Error

This figure shows the accumulated absolute error of each Heart Cell-DEVS simulation with the different quantums.
The references are:

DEVS is the non quantified simulation.
DEVS-q is the simulation with standard quantum.
DEVS-q-y is the simulation with standard quantum and dynamic ratio strategy 1.
DEVS-q+Y is the simulation with standard quantum and dynamic ratio strategy 2.
DEVS+Q is the simulation with quantum hysteresis.
DEVS+Q-y is the simulation with quantum hysteresis and dynamic ratio strategy 1.
DEVS+Q+Y is the simulation with quantum hysteresis and dynamic ratio strategy 2.

The value on the right of the "y" or "Y" strategy indicator is the ratio used for the dynamic quantum.
As can be seen on the figure, the lowest absolute error was obtained with the dynamic quantum strategy 1 with ratio 0.9 (Hysteresis or Standard): DEVS+Q-y0.9 and DEVS-q-y0.9. The line of the standard can't be seen because of overlapping with the Hysteresis. With the same quantum but strategy 2, the error is the biggest of this set of tests. No differences can be observed for this model between standard and hysteresis quantums. This is reasonable, because of the nature of Hysteresis quantum, which differs of standard when direction changes are present and on this model (Heart) there are no significant direction changes (only one at value +24 aprox as we can see on the figure of the Heart model).

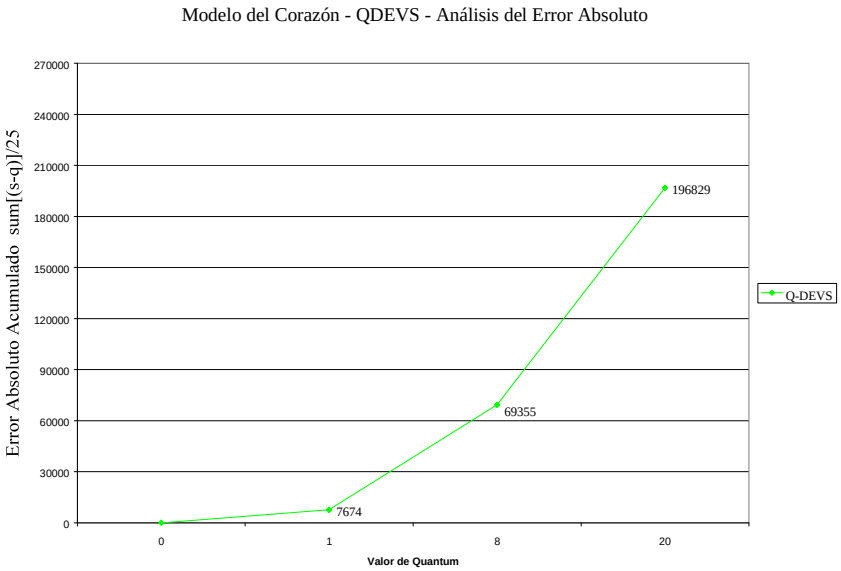


Figure 35. Q-DEVS Heart model Absolute Error

On Figure 35 (with Adaptive Quantization Q-DEVS) we can see that absolute error is the same as standard quantization (DEVS-q line on Figure 34). For instance, with quantum 8, both absolute errors (standard and adaptive) are 69355. This model works with the same quantum values and with the same function of Cell-DEVS. This model has no dependencies between neighbors values and that is the reason because the error has no difference with respect to the standard quantum. The accumulated error is produced because of the individual cell differences with the non quantified model. The difference of a cell has no impact on the other cells (the neighbors) because of the simplification mentioned before (no dependency with the neighbors).
As we can see, with internal quantum value "0" this mechanism works exactly as the original model without quantum. That is because with quantum "0" the inverse function of the original one makes no difference and returns the default delay and the next value for the cell without making any additional calculation or changes.

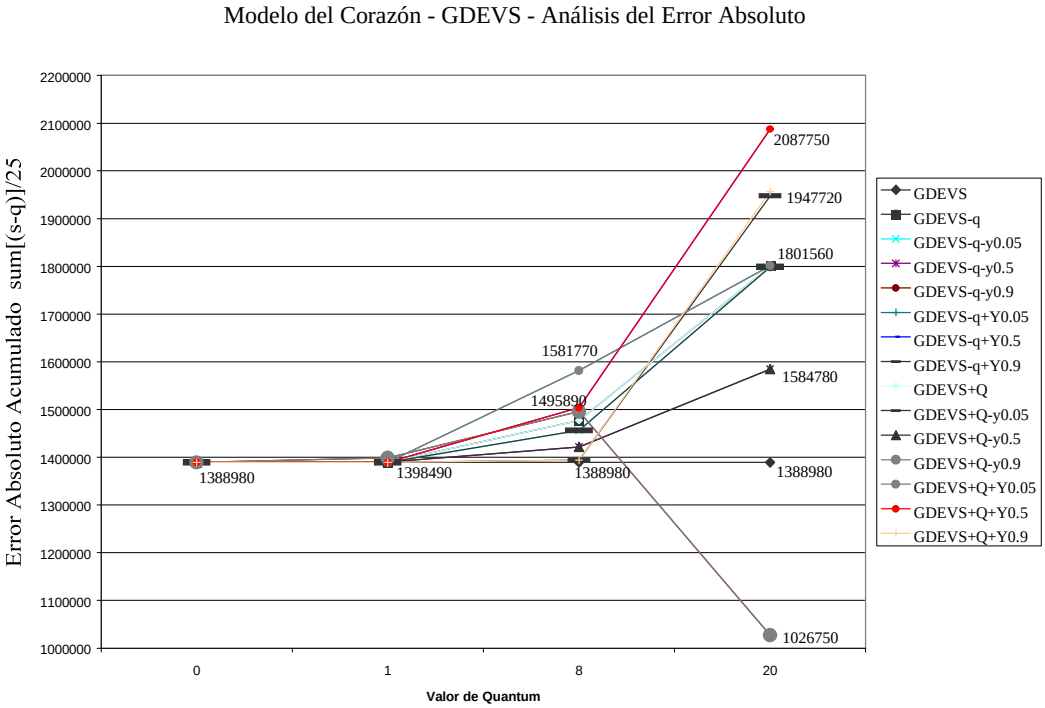


Figure 36. GDEVS Heart model Absolute Error

With GDEVS, error is bigger, on the order of 1.000.000. Also GDEVS quantized simulations where done to evaluate behavior. GDEVS with quantum Hysteresis (and Standard) 20 and dynamic strategy 1 ratio 0.9 (GDEVS+Q-y0.9 and GDEVS-q-y0.9) has some enhancement of the absolute error (remember that this is a GDEVS model approximated with polynomial functions where quantum 0 has already an error within). On this case, if we make a more complex and accurate polynomial approximation, the error can be reduced while the model design will be more complex and hard to evaluate. E.g. if the model has n-variations in the simulation and we define n order 0 polynomials (with all the constant values) for each instant we will have no error with respect to the original model but the design will be very complex (and large, because we'll have n polynomials where n is the number of different values of the model), which looks as a cellular automata with n-states (one for each value). In the other hand, if we define only one polynomial, the model design will be very simple but the incurred error will be high (because one value will represent the complete model). On this case we have 12 polynomials. On the next analyzed model (Watershed, which has only one order 1 polinomy) we experimented this behavior.

Relative Error analysis

The relative error is analyzed for each simulation and the figures shows the incurred error with each quantum value.

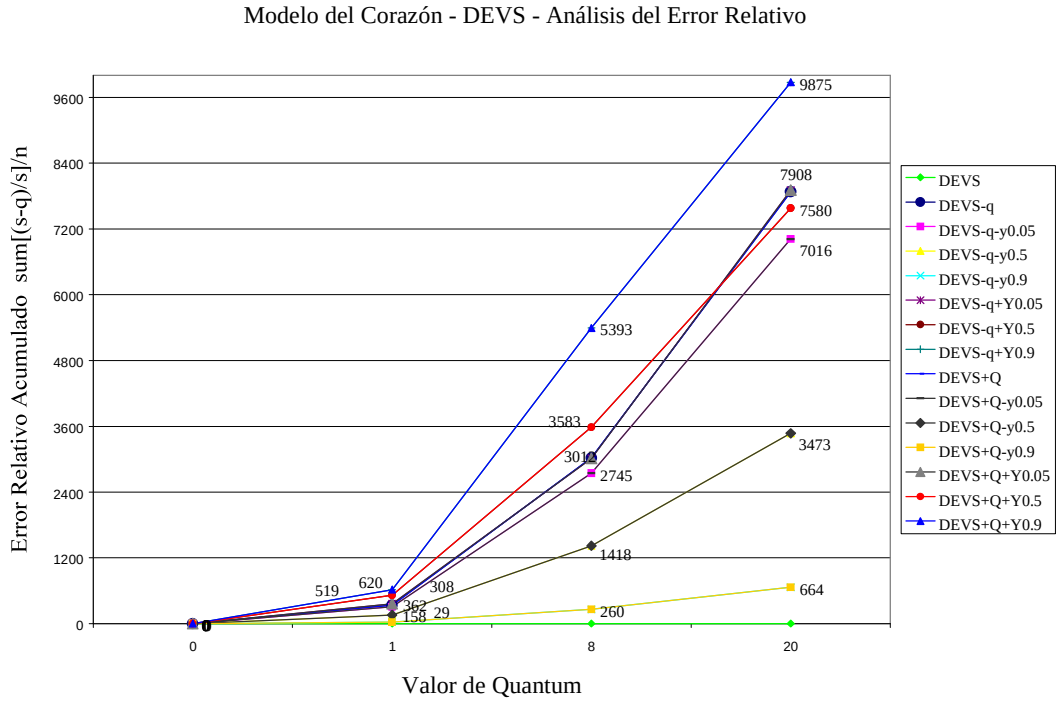


Figure 37. Cell-DEVS Heart model Relative Error

The relative error follows the same tendency of the absolute error except for quantum 20 dynamic strategy 2 ratio 0.5 (DEVS+Q+Y0.5), which in this case, the relative error is lower than strategy 1 (yellow line).
Next Figure 38 shows the relative error of Q-DEVS strategy.

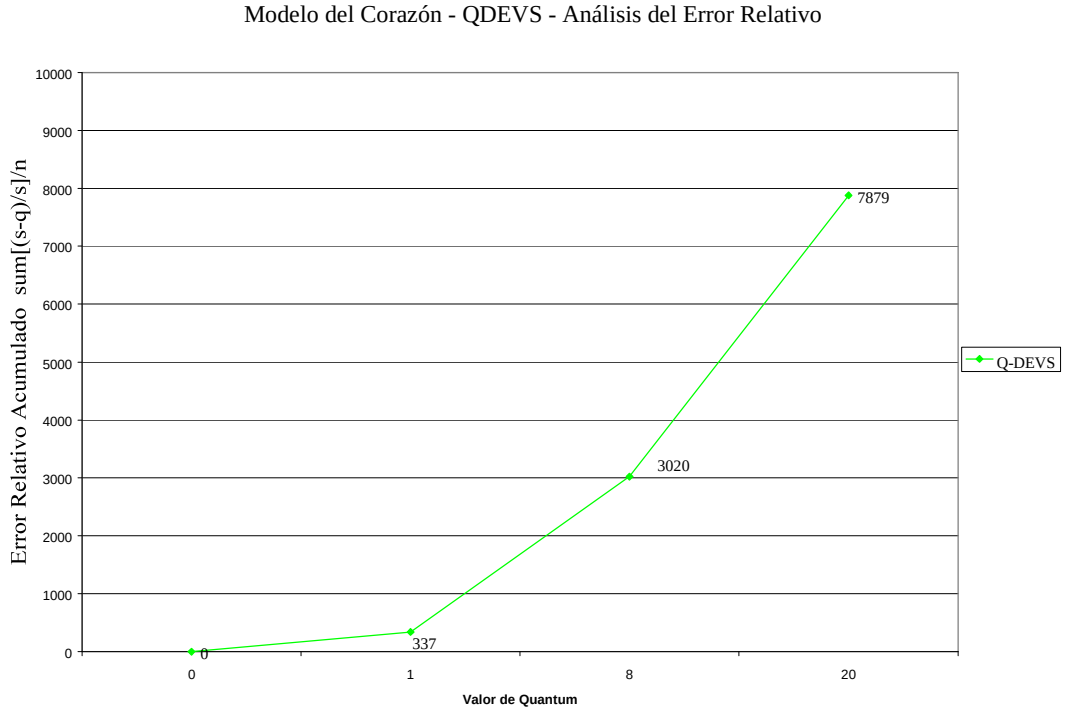


Figure 38. Q-DEVS Heart model Relative Error

On figure 38 (Q-DEVS relative error) we have similar comparisons done for absolute error. The values are the same with Q-DEVS and with standard quantum. This is because the model was implemented with

no dependency of the neighbors to calculate the local value. On others models with interaction of the different cells this can be different and Q-DEVS simulations will have different results.
Next figure 39 shows the GDEVS relative error analysis.

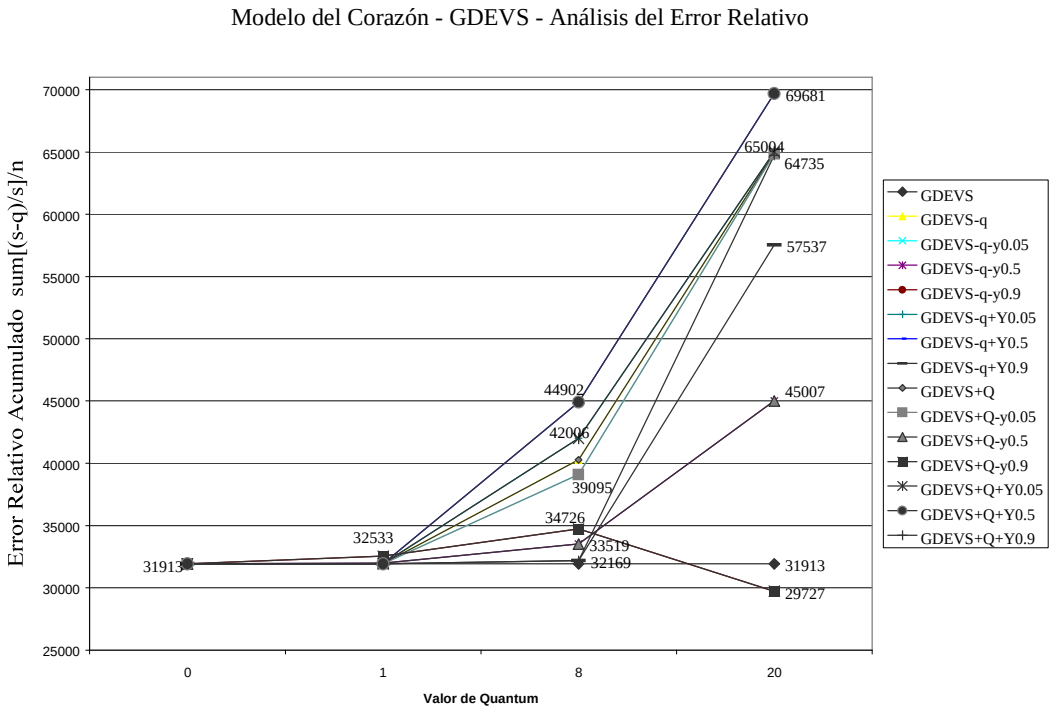


Figure 39. GDEVS Heart model Relative Error

With GDEVS, relative error is bigger than other methods but there we have a better result with GDEVS quantified with dynamic quantum strategy 1 ratio 0.9. The GDEVS simulation with no quantum has already an error within because of polinomy approximation. Adding dynamic quantum strategy 1 ratio 0.9 the relative error was smaller because of a smaller difference with the original numbers when quantifying the GDEVS values (bigger values produced by GDEVS polynomials were minimized with the dynamic quantum). This was an exceptional and very particular result on this specific model.

Messages analysis

Message analysis was made comparing the number of generated messages on the simulations (considering the simulation end time of the smaller simulation, because with some quantum values and types, the simulation can finish before the programmed ending time and the comparison will be not valid if we don't consider the ending time of the smaller).
The analyzed messages are the total number of messages (this includes initialization, internal, external, output and done messages). The output messages were analyzed separately.

Total Messages analysis

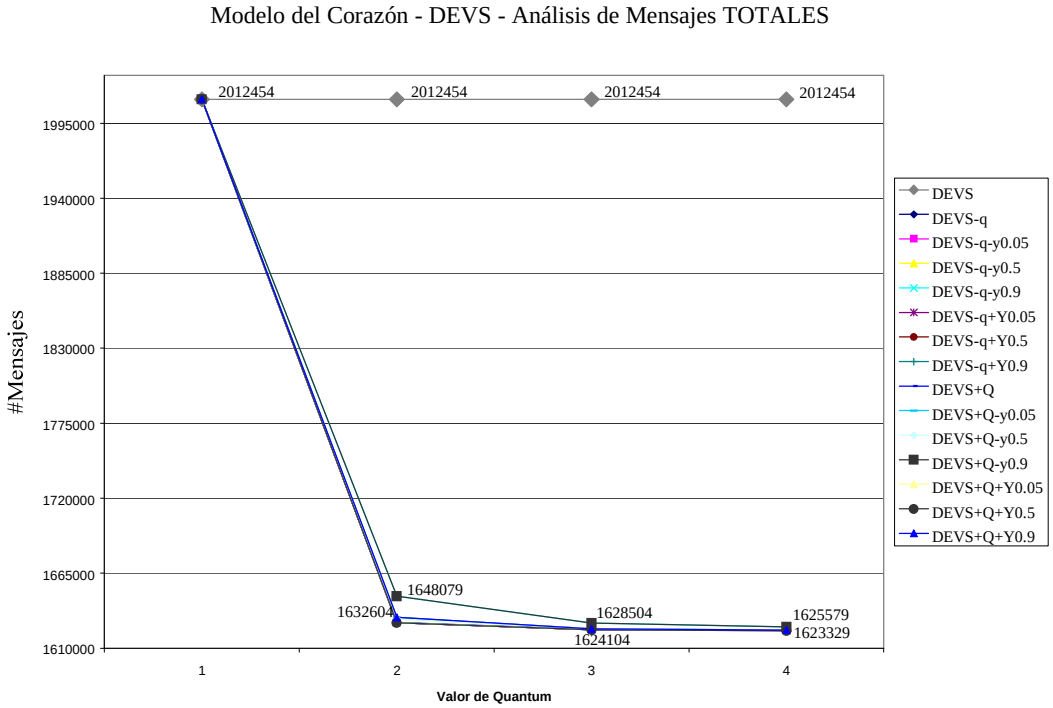


Figure 40. Cell-DEVS Heart model Total Messages

The total number of messages with this type of quantum has a maximum reduction of 20% compared with the simulation without quantum. This is because the simulation continues instead of no output is generated. That's the reason because we will analyze the output messages separated. Those are the messages to be highly reduced while quantifying with the different combinations of quantum types and mechanisms.

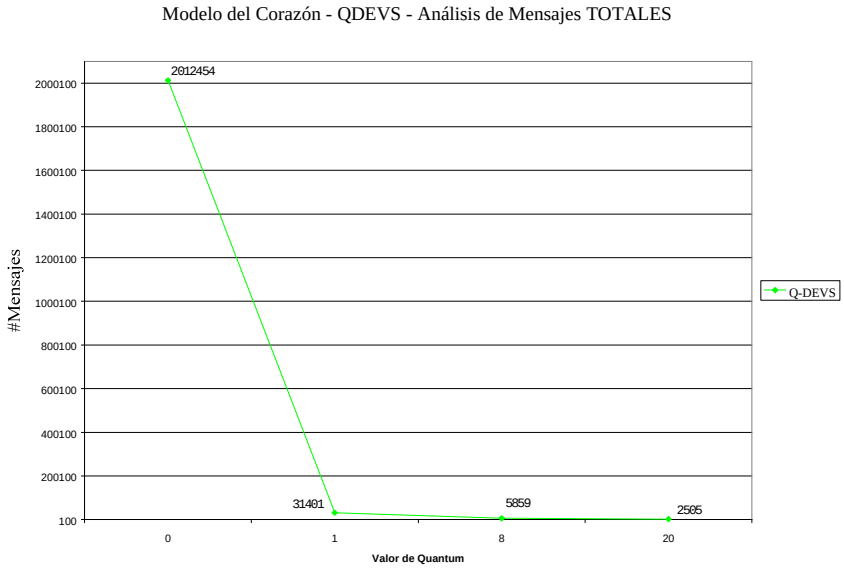


Figure 41. Q-DEVS Heart model Total Messages

On this case (Figure 41), with adaptive quantization, the total number of messages was reduced significantly. The maximum reduction with quantum 20 it's 99.87%. With this technique, the simulation will be automatically programmed on the next time a value change its region and that is the advantage of this quantum technique, because the model is not generating internal and external messages every x ms, it's only generating messages when quantum region changes takes place. This simulation is not doing a

step-by-step simulation advance time as in the standard Cell-DEVS quantization and that's the reason because the biggest message number reduction.

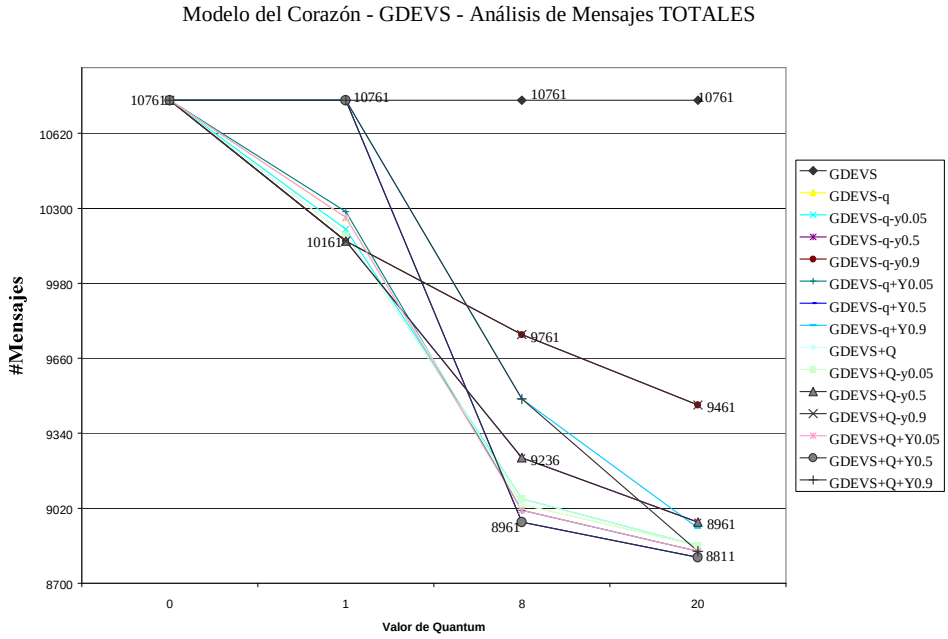


Figure 42. GDEVS Heart model Total Messages

In this case, with GDEVS methodology, the number of total messages are also significantly reduced (compared with the original simulation of Cell-DEVS, who has 2.012.454 messages and here we have less than 11.000 total messages).

The message reduction with GDEVS (with no quantum) is about 99.4%. With quantized GDEVS, reduction is a little better (99.6%). Here also, the reduction is significantly because the model is not generating events every fixed x ms, it's generating messages only when the polynomi changes it's values, depending on the state of the cells.

Output Messages analysis

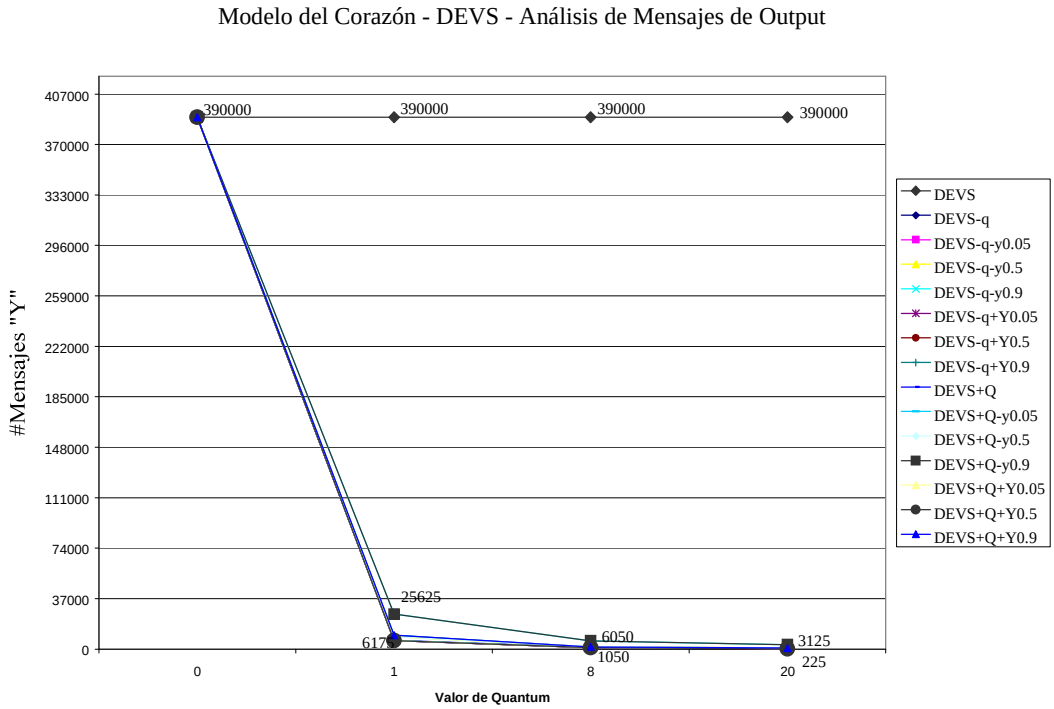


Figure 43. Cell-DEVS Heart model Output Messages

As we can se on figure 43, output messages are reduced significantly with quantization, to the extreme of having 225 output messages on the quantified model and 390.000 output messages on the non quantified model (99.95% of reduction). Of course, while message are reduced, potential error is incurred. Here there is a significant difference between dynamic strategy 1 ratio 0.9 and the other quantum values (reduction it's a 76% lower with dynamic strategy 1 ratio 0.9). On the other hand, while analyzing the error, we have a lower error with dynamic strategy 1 ratio 0.9 (more messages but lower error).

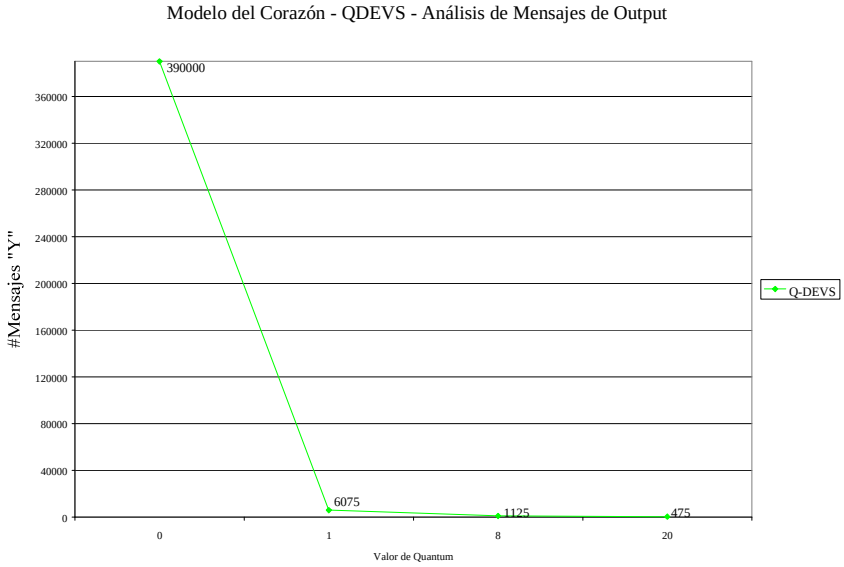


Figure 44. Q-DEVS Heart model Output Messages

With adaptive quantization (Figure 44 Q-DEVS), difference is not too big as the comparison of total messages, because output messages are reduced always (with both techniques). In the total messages we have a big difference between Q-DEVS and standard quantum, but when output messages are analyzed, both reduces significantly the output messages because these are the optimized messages with both techniques but with Q-DEVS, all messages (output and every message) are reduced because the cell next event time is managed, not only the local updating of the cell (like on the standard quantum).

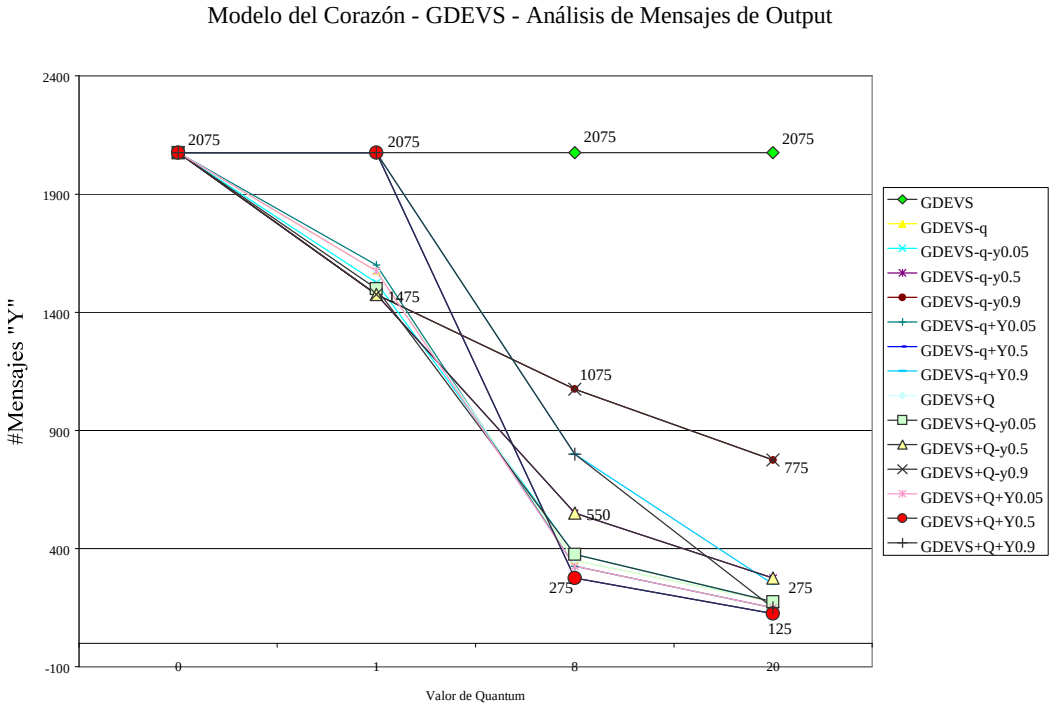


Figure 45. GDEVS Heart model Output Messages

In this case (figure 45 GDEVS methodology) the number of output messages behavior is similar to total messages behavior (compared with the original simulation of Cell-DEVS, who has 390.000 output messages). The message reduction with GDEVS only (without additional quantum) is about 99.47%, but if we see the quantized GDEVS, reduction is bigger (on output messages) than with total messages, because (as mentioned before) output messages are significantly reduced (about 99.97%) with the quantization techniques (standard and hysteresis) combined with GDEVS.

Simulation time analysis

This section shows the simulation time analysis of each technique. The analyzed time is the real time the simulations takes (from the model loading until the simulation ended) in terms of CPU usage time. The time was measured in seconds (without milliseconds). This analysis is relative to the computer used (CPU speed, data BUS, disk, memory, etc.) but all simulations were made on the same computer to allow valid time comparisons analysis.

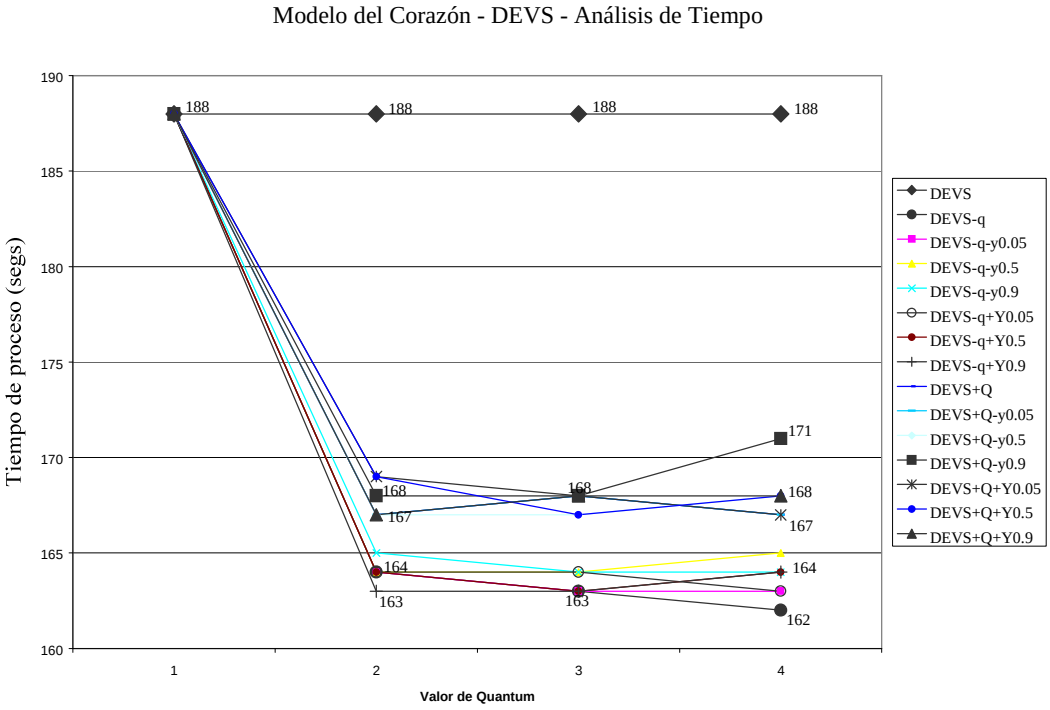


Figure 46. Cell-DEVS Heart model Processing Time

As we can see on figure 46, the total simulation time without quantum with the Cell-DEVS implementation was 3 minutes 8 seconds. The best time obtained with standard and hysteresis quantums (also with all the combinations of dynamic quantums) was 2 minutes 40 seconds (14% lower than the original simulation). The best time was obtained with standard quantum (no dynamic strategies). The variations are not too significantly with this technique.

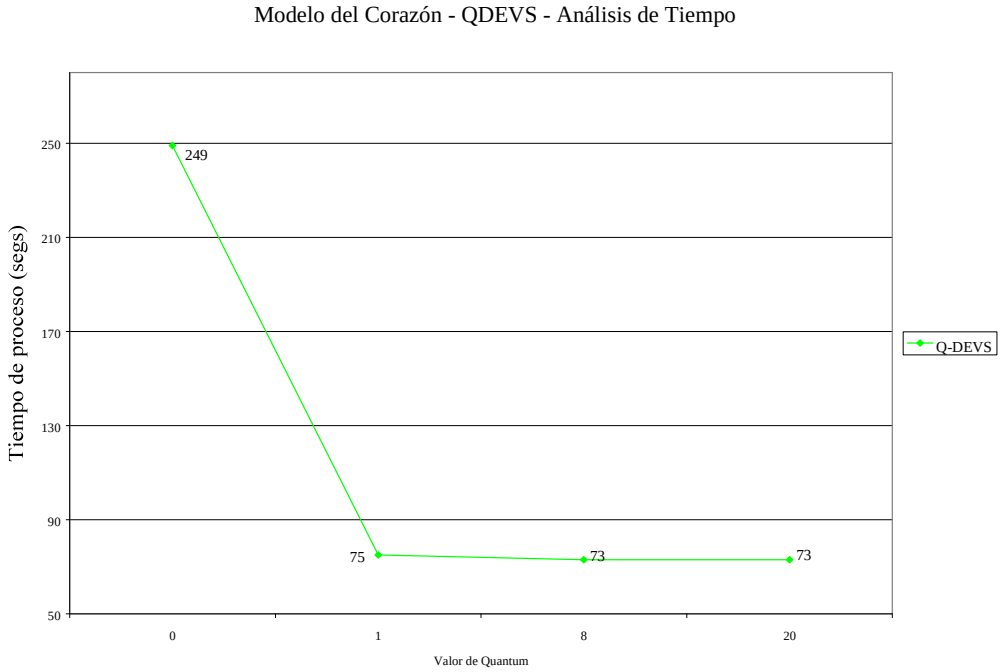


Figure 47. Q-DEVS Heart model Processing Time

Simulation time with Adaptive quantums (Q-DEVS) without quantum (quantum 0) is bigger (4' 9", 32% bigger) than the original model (3' 8"). When no quantum is used on Q-DEVS we have the same simulation of the original one, but using Q-DEVS technique with quantum 0. When a Q-DEVS quantum value is specified, the simulation time reduces significantly, up to a 60%. The best time here was 1 minute 13 seconds, which is a 62% lower of the simulation time of the original model. The simulation takes more time with Q-DEVS quantum 0 because the function is more complex than the standard function, because of additional calculations made with the inverse function, but when Q-DEVS quantum value is specified, the complexity time overhead is insignificant because the total time reduction is bigger.

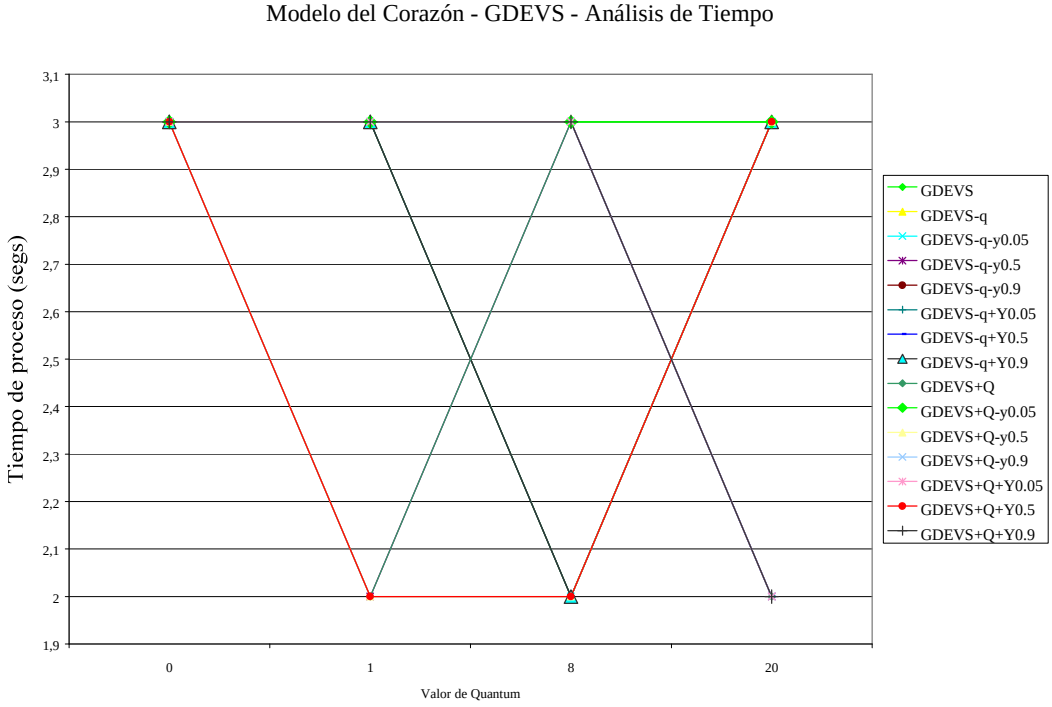


Figure 48. GDEVS Heart model Processing Time

On Figure 48 we can see the simulation time of GDEVS. Time is the smallest of all the techniques (less than 3 seconds). It varies between 2 and 3 seconds compared to the 188 seconds of the original Cell-DEVS simulation. Reduction is more than 99%, with GDEVS with or without additional quantum.
The main difference here is that the local computation function has a low complexity and the messages involved are less than the other models (2.000.000 messages vs. 10.000) and this results on a faster simulation because of the small number of generated messages.

7.3 Watershed Model

The Watershed model was tested using the following selected quantums types and values, showed on the tree figure below (figure 49).

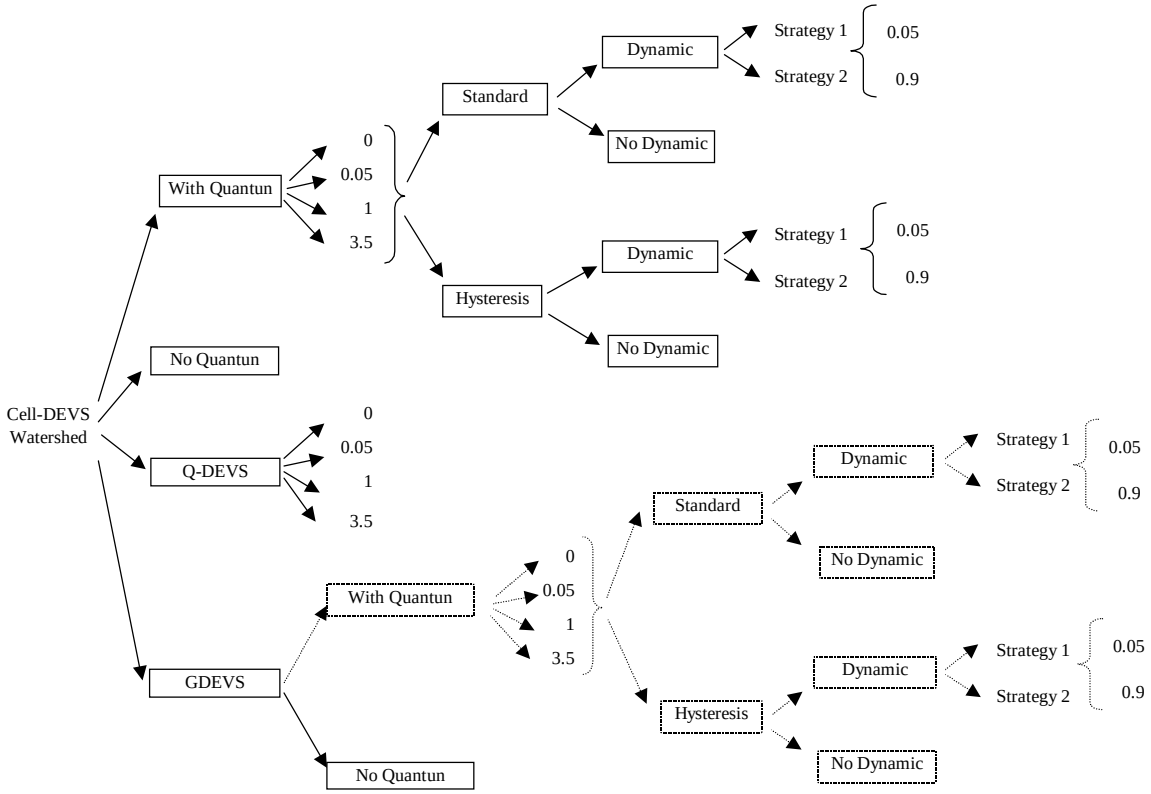


Figure 49. Quantum types and techniques used with Watershed model

The description of this values is the same as in the Heart model, but the values are different, according to this model, which has a lower range of values (that implies using lower quantum values).
This model was tested using the land topology showed on figure 50, which means that all watershed simulations were originated form this initial state. Figure 50 shows the initial state of the land before watershed events.

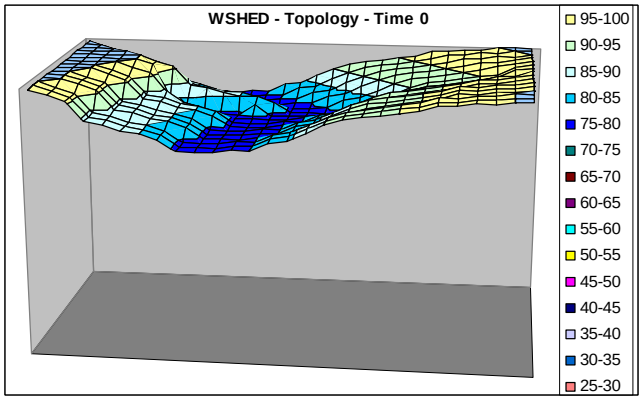
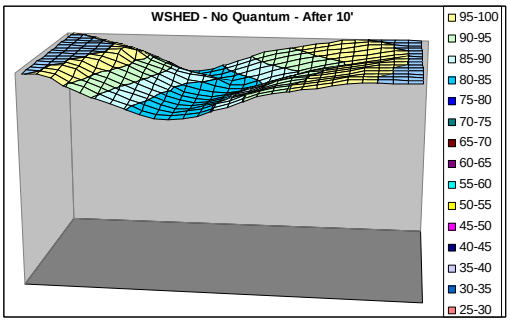


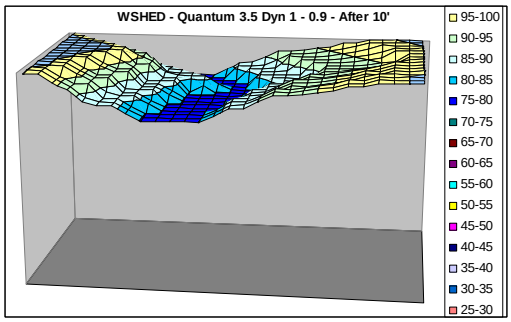
Figure 50. Initial Watershed state

Figure 51 shows simulation examples of the graphical output, were we have the status of the land after 10' of rain. The land surface can be observed for different simulation strategies and quantums:

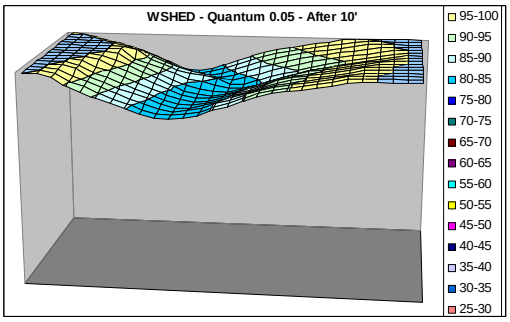
No Quantum
Messages: Y= 268800, All= 1.347.300



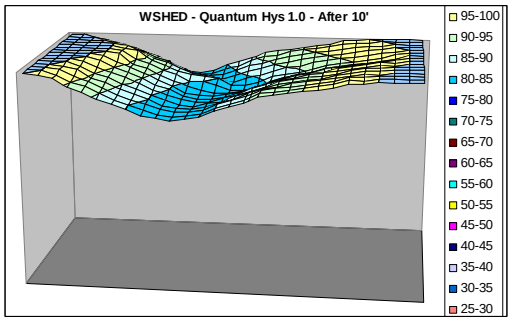
Quantum Standard 3.5 Dynamic 1 - 0.9
Messages: Y= 4021, All= 1.082.521
Abs Error: 92, Relative Error: 1



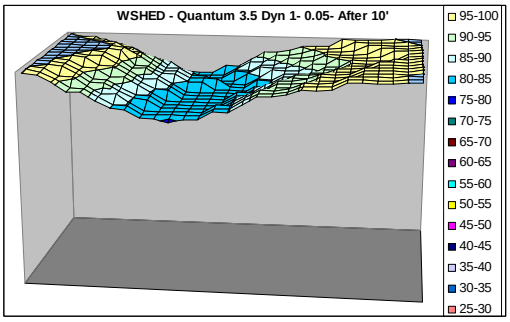
Quantum Standard 0.05
Messages: Y= 27.972, All= 1.106.472
Abs Error: 12, Relative Error: 0.14



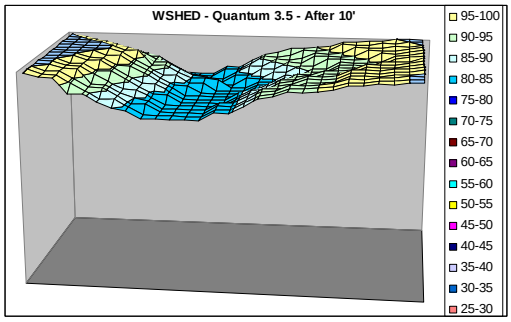
Quantum Hysteresis 1.0
Messages: Y= 1.188, All= 1.079.688
Abs Error: 258, Relative Error: 2.8



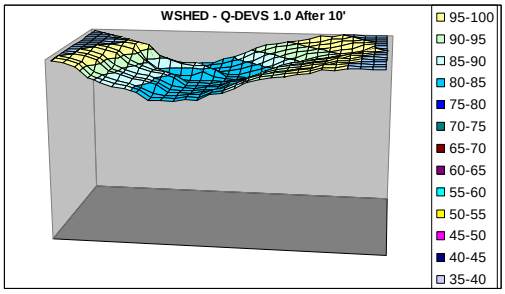
Quantum Standard 3.5 Dynamic 1 - 0.05
Messages: Y= 438, All= 1.078.938
Abs Error: 474, Relative Error: 5



Quantum Standard 3.5
Messages: Y= 362, All= 1.078.862
Abs Error: 482, Relative Error: 5



Q-DEVS 1.0
Messages: Y= 536, All= 4.956
Abs Error: 559, Relative Error: 6.2



GDEVS
Messages: Y= 11.610, All= 66.390
Abs Error: 512, Relative Error: 5.8

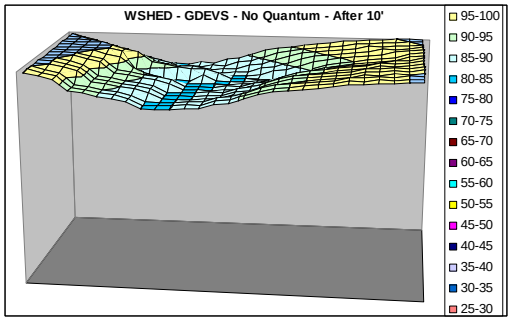


Figure 51. Watershed Result Simulation Examples

On the above figures, some graphical differences can be observed because of incurred error. The quantum type and value, the number of output and total messages and the accumulated relative and absolute error is showed for each graph.

Error analysis

This section analyzes the error (absolute and relative), which was calculated as showed on the Heart model (with the same formulas and tools).

Absolute Error analysis

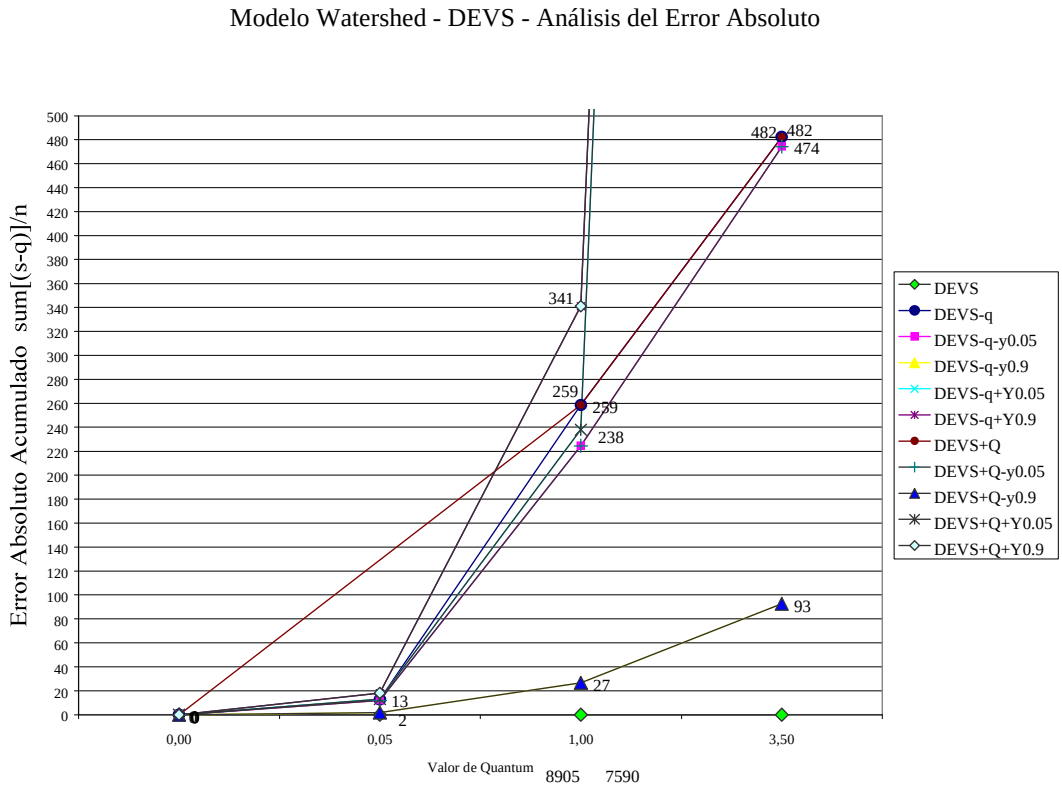


Figure 52. Cell-DEVS Watershed model Absolute Error

As we can see on figure 52, on this model, the absolute error is bigger while using dynamic quantum strategy 2. With dynamic quantum strategy 1 ratio 0.9 we've got the lowest absolute error. This is reasonable here, because this model has interaction between its cells (it is different than the Heart model, where all the cells were independent). As we can see, dynamic quantums gets better results on this model (80% less absolute error).

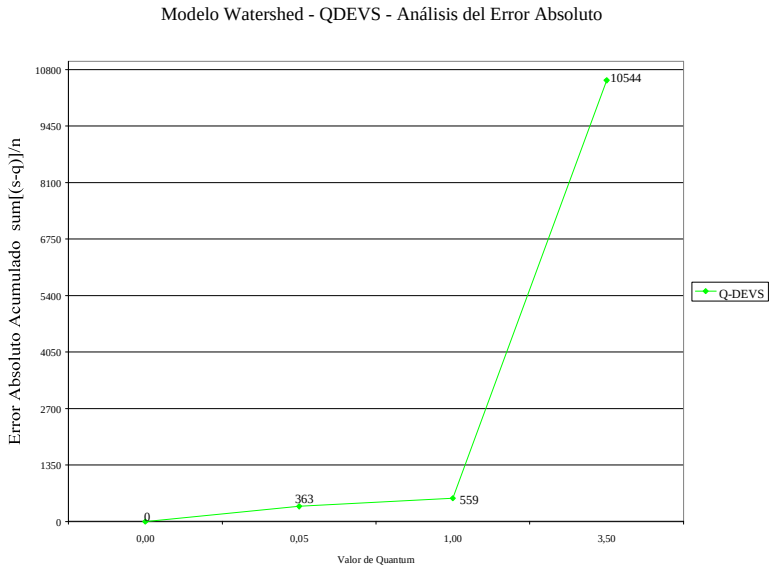


Figure 53. Q-DEVS Watershed model Absolute Error

On this model, Q-DEVS has not better results than some standard quantums. For instance, with Q-DEVS quantum 1.0 we have an absolute error of 559 and with DEVS standard quantum 1.0 we've got 259. This is also because of the non independent cell behavior described before. With the prior model (the Heart model) we had the same ratio of error between DEVS and Q-DEVS because the cells were independent, but here, cells has to receive and send water to its neighbors in order to simulate watershed, and this makes the difference on this model using Q-DEVS. Q-DEVS is not interacting with the neighbors while calculating the next value and delay for a cell because the inverse calculation is not made at future simulation times with all the future neighbors values for each cell while trying to anticipate the value for a given quantum, so the value is calculated with "old" neighbors values which result on a bigger error while using this type of quantum.

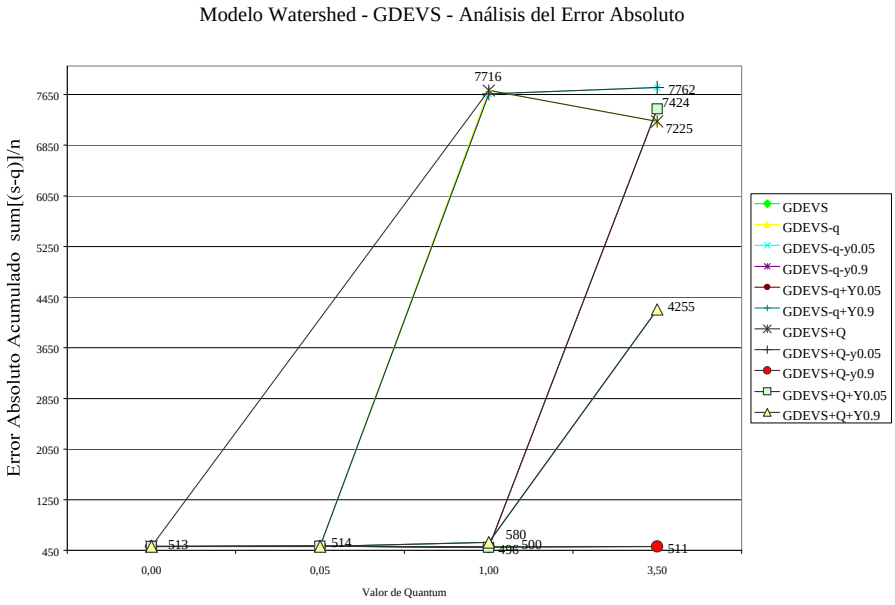


Figure 54. GDEVS Watershed model Absolute Error

On the GDEVS simulation without quantum, error is similar to standard quantum (no dynamic) 3.5. GDEVS with quantum 1.0 and dynamic strategy 1, has a little lower absolute error than base GDEVS. This is because the GDEVS simulation without quantum has an error within the model design because of the polynomi approximation and the quantization (the same effect described on the Heart model).

Relative Error analysis

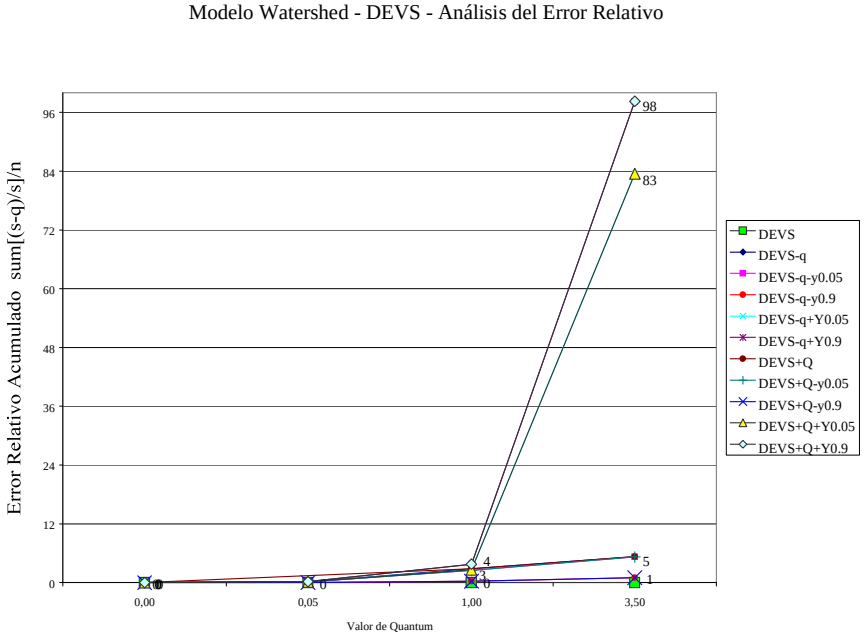


Figure 55. Cell-DEVS Watershed model Relative Error

Relative error is small with dynamic quantum strategy 1 ratio 0.9 for this model (same result we've got with absolute error).

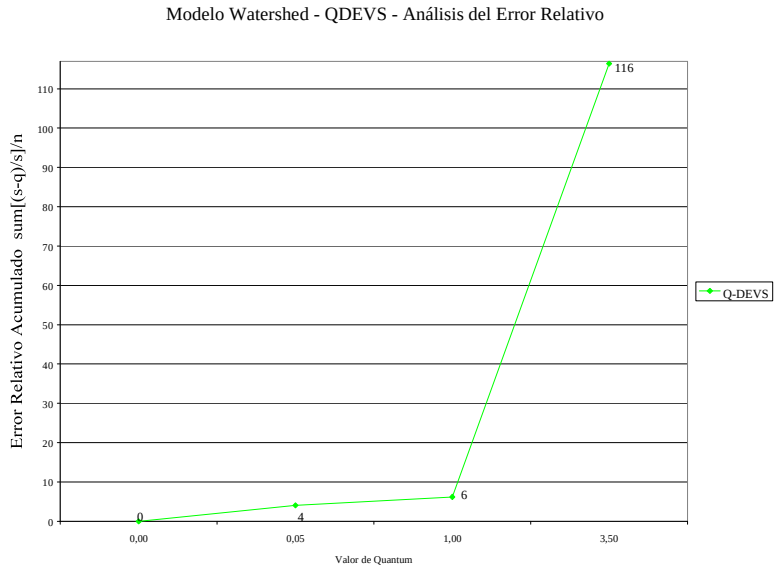


Figure 56. Q-DEVS Watershed model Relative Error

Q-DEVS with quantum 0.05 and 1 has a smaller relative error considering the important reduction of messages and simulation time obtained with this technique. Here the relative error is about 6 (see figure 56) and with the standard quantum we've got around 4 (see figure 55).

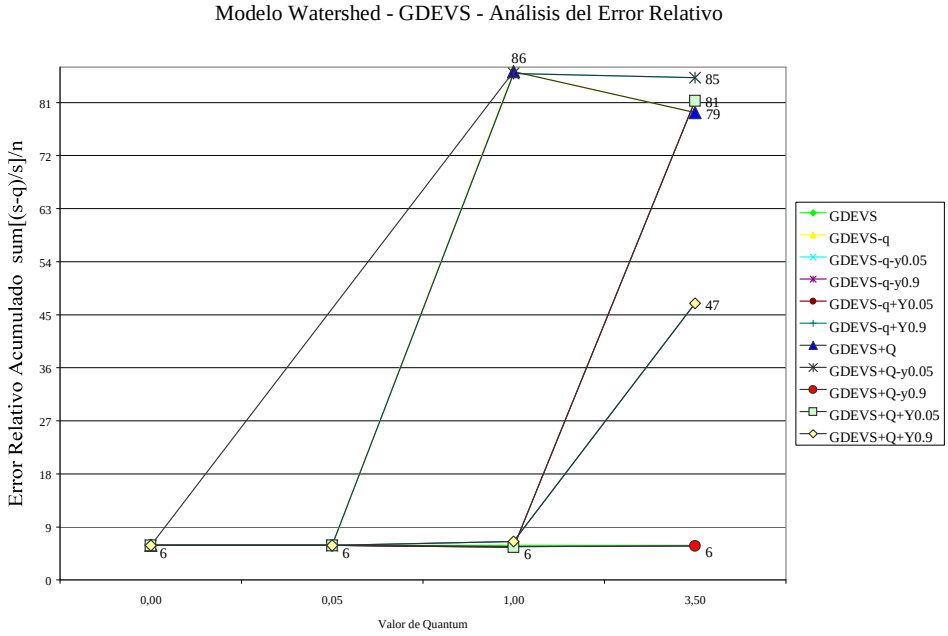


Figure 57. GDEVS Watershed model Relative Error

Also with GDEVS (without quantum and with quantum strategy 1) we've got similar relative errors (compared with the standard quantums). With GDEVS strategy 1 relative error is approximately 6 (figure 57) as we've got with standard quantum (figure 55).

Messages analysis

On this section we can see the output and total messages analysis made for the Watershed model. The analysis type is the same used for Heart model.

Total Messages analysis

Here are the total number of messages generated with the different simulations. Important differences can be seen with different quantums values and strategies.

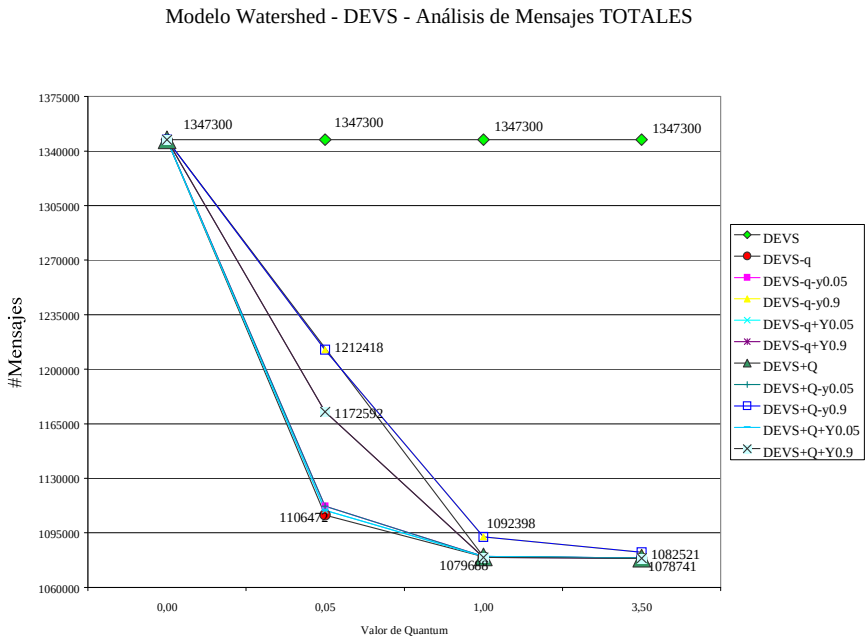


Figure 58. Cell-DEVS Watershed model Total Messages

With Cell-DEVS quantums, the total number of messages are reduced about 25% (maximum). With big quantum values for this model (3.5 is a big quantum value because of the variation of this model is smaller than 3.5 most times) the number of messages are reduced in the same order (25%) because with the cell interaction the simulation is generating more external messages to keep the model alive when quantifying.
On the next graph (figure 59), we will see that the total messages are significantly reduced because of the adaptive quantization, which works with a different technique more effective for the total number of messages.

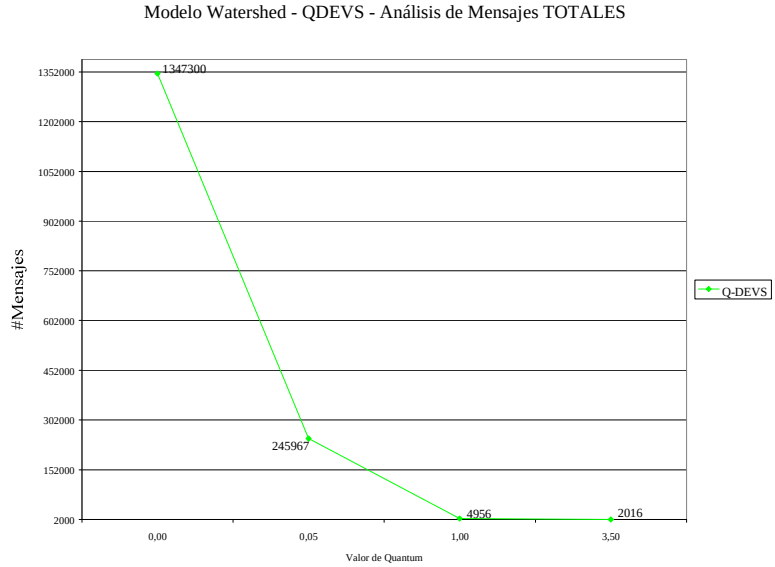


Figure 59. Q-DEVS Watershed model Total Messages

On figure 59 we can see the result of quantization with the consistent reduction of messages. Of a total of 1.347.300 messages, with an small quantum value (0.05) the messages were reduced to 245.967 (more than an 80%). With a quantum value of 1.0, the messages were reduced up to 99%. Of course, the incurred error is bugger when more messages reductions we have.

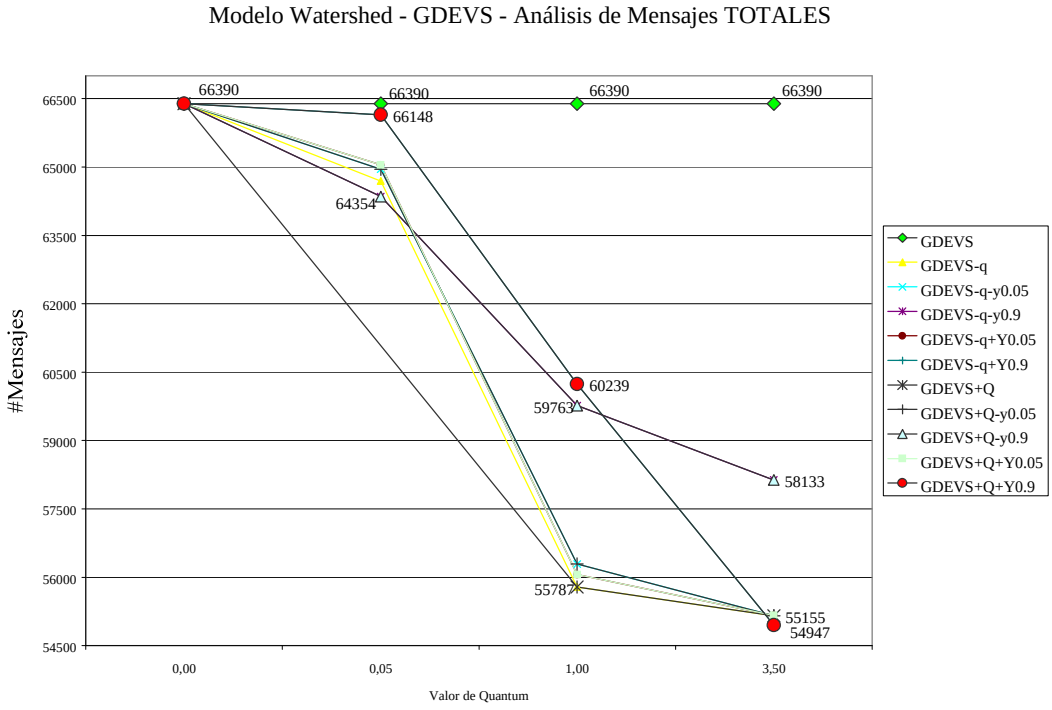


Figure 60. GDEVS Watershed model Total Messages

On figure 60, as a difference with the Heart model, GDEVS has more messages than Q-DEVS while no very significant error differences can be observed between them.

Output Messages analysis

The lowest number of output messages was obtained with the Q-DEVS strategy.
With GDEVS, there are more messages than with Q-DEVS and also more than some standard and hysteresis quantums (except with dynamic strategy 1). E.g., with quantum standard 1.0 (no dynamic strategy) we have between 743 and 1.188 output messages. With Q-DEVS quantum value 1.0 we have 543 output messages and with GDEVS we have 11.600 messages.
Of course, as mentioned before, with DEVS, the external messages were incremented because of the absence of output messages (when no output messages are generated, no events are scheduled for the evaluated cell and its neighbors and the simulation can stop if there are no events, but automatically external events are generated to advance the simulation time with the new "-c" argument added to the tool.
Figure 61 shows the output messages generated with standard, hysteresis and dynamic combination strategies.

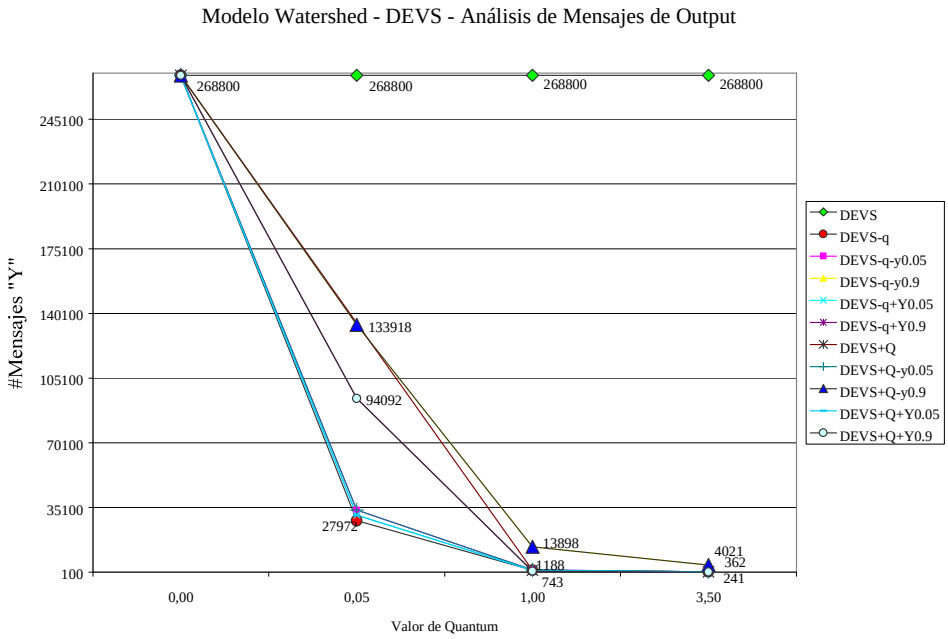


Figure 61. Cell-DEVS Watershed model Output Messages

On the next graph (figure 62 - Q-DEVS quantization) the messages are reduced in the same proportion of the DEVS quantization because we are comparing here the Output messages, which are reduced significantly with an small quantum value for this model (considering that the output messages are reduced significantly compared with the total messages).

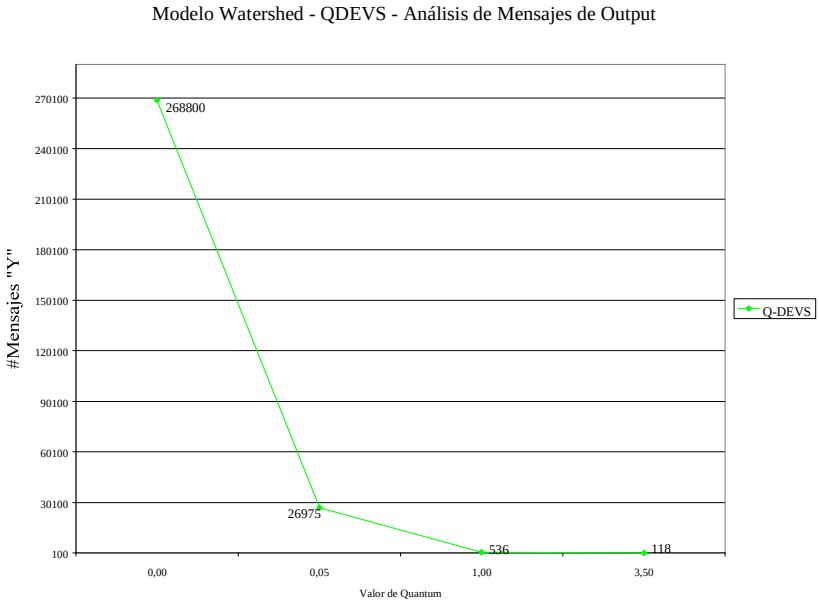


Figure 62. Q-DEVS Watershed model Output Messages

On figure 63 (GDEVS simulation) we can see the output messages reduction on the same proportion of the total messages for the GDEVS simulation. Of course, GDEVS has already a big reduction on the total messages and that reduction involves also the output messages. On the GDEVS case (as in the Q-DEVS) no more extra external messages (X) are generated because of the working mode of this techniques (both are not using the CD++ quantization, it uses an internal quantization on the design of the model). On the Cell-DEVS simulation, we can see that there is a big difference between total and output messages when quantifying, because it uses the CD++ generic quantization, which generates additional messages to keep the model alive.

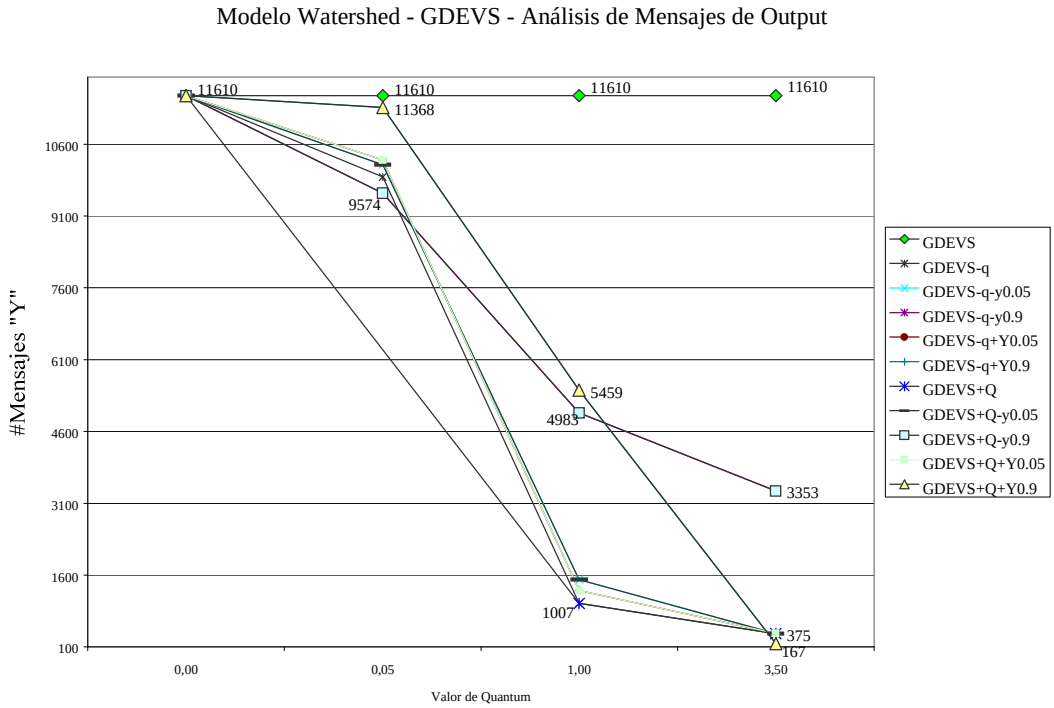


Figure 63. GDEVS Watershed model Output Messages

Simulation time analysis

Simulation time is not significantly reduced with Cell-DEVS quantums on this model (130 seconds is the maximum and 111 is the best time) but with Q-DEVS we have (with quantum value 1.0) 8 seconds of simulation time and with GDEVS between 5 and 7 seconds. Next three figures (figure 64, figure 65 and figure 66) show the simulation time for the different techniques and quantum type an values used.

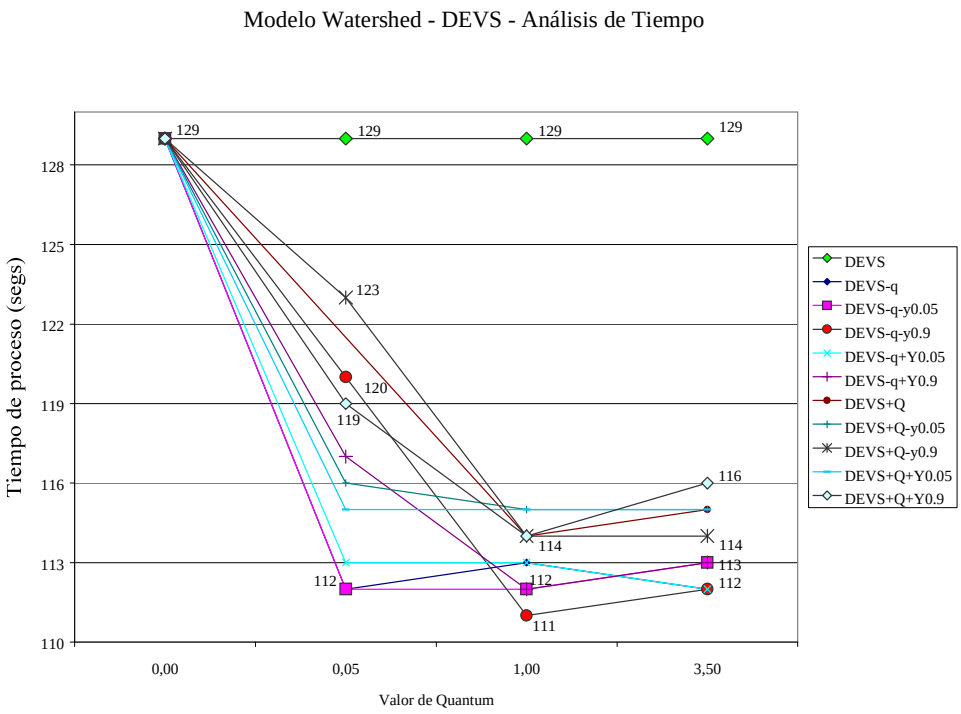


Figure 64. Cell-DEVS Watershed model Processing Time

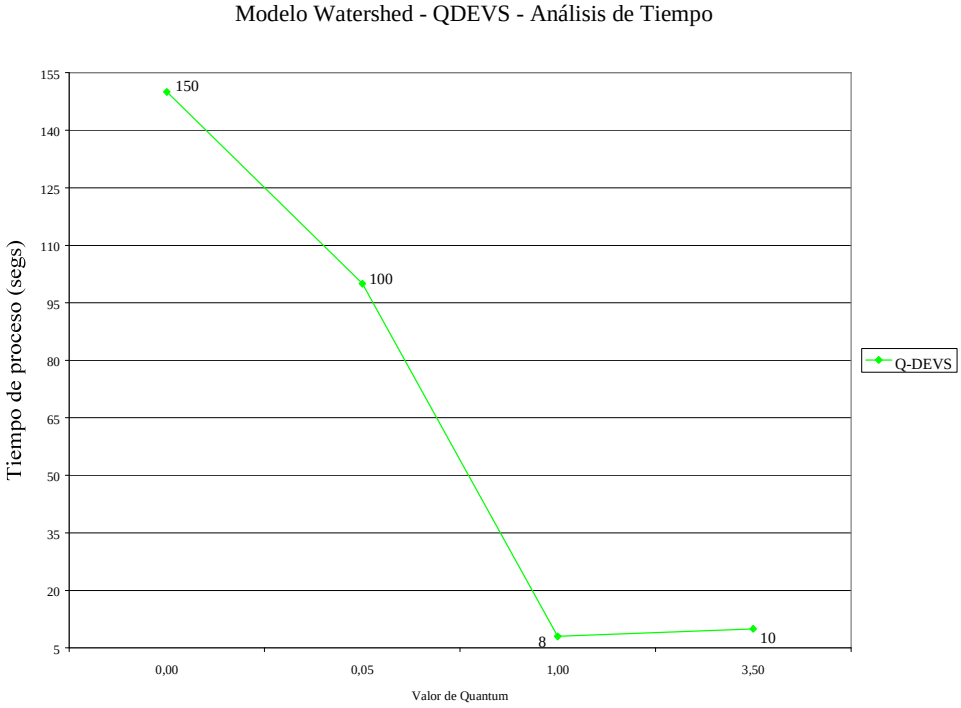


Figure 65. Q-DEVS Watershed model Processing Time

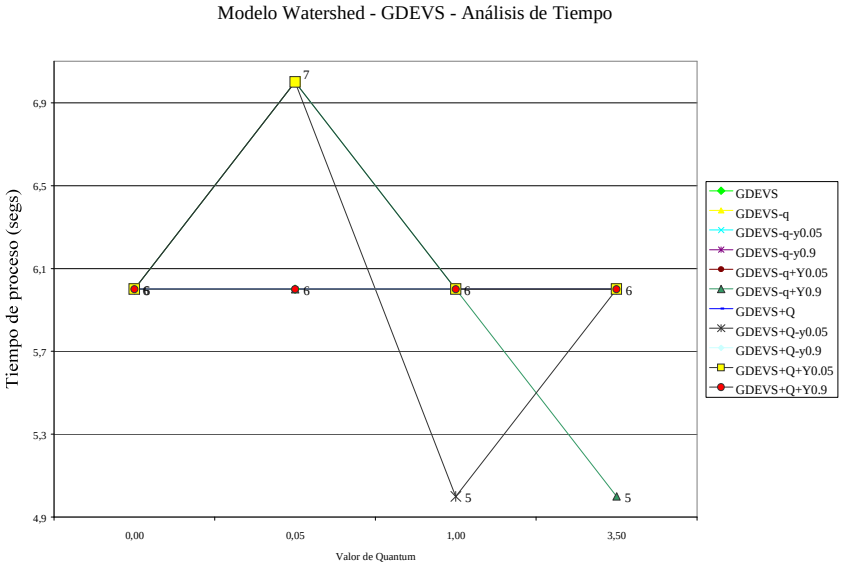


Figure 66. GDEVS Watershed model Processing Time

7.4 FIA Model

For this model, all quantum types and strategies were analyzed except for GDEVS, which was not implemented because of the behavior characteristics of the model. This model has a "translation" and a "diffusion" function, which alternate the cell values. The behavior of its cells is not uniform. As we will see on the function figures, each cell has a completely different behavior. To implement the GDEVS technique for this model, we would have needed to define a different polynomi for each cell (but this was not made on this work).

Figure 67 show the different quantum combinations used with the FIA model.

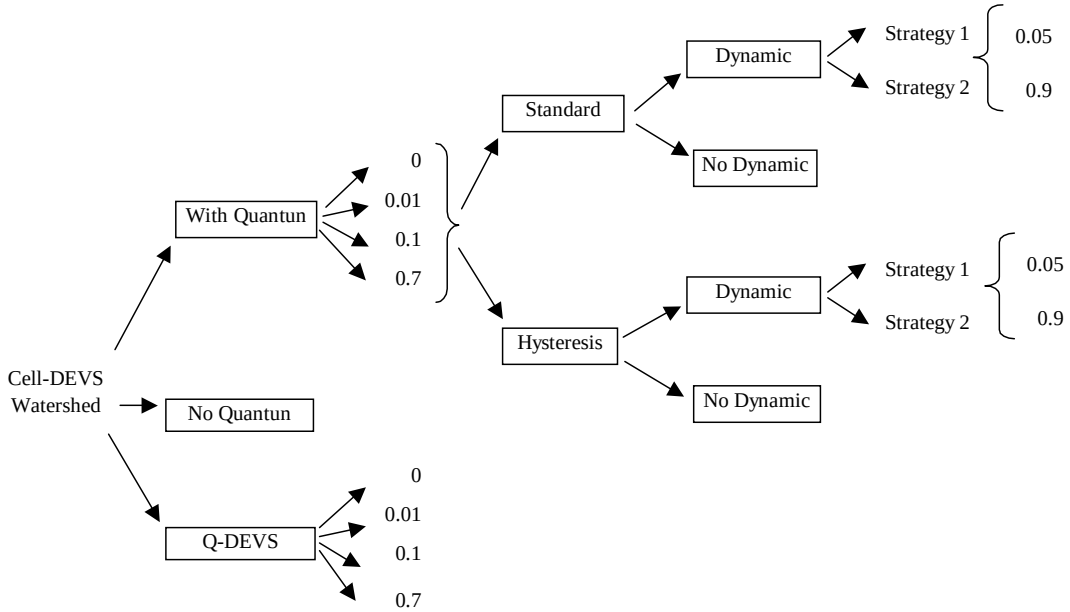


Figure 67. Quantum types and techniques used with FIA model

On Figure 68 we can see some simulation pictures at specified times, after the sample was introduced on the tube in order to simulate the flow analysis. Some figures with different quantum values and techniques are shown and some graphical disturbs can be observed on the quantized examples.

These are some simulation examples of the graphical output:

No Quantum	**Quantum Standard 0.7**, 120ms
Initial - time=0, Sample –white- injected	Messages: Y= 15.706, All= 7.463.348
Messages: Y= 970.892, All= 8.357.282	Abs Error: 11.732, Relative Error: 82.123

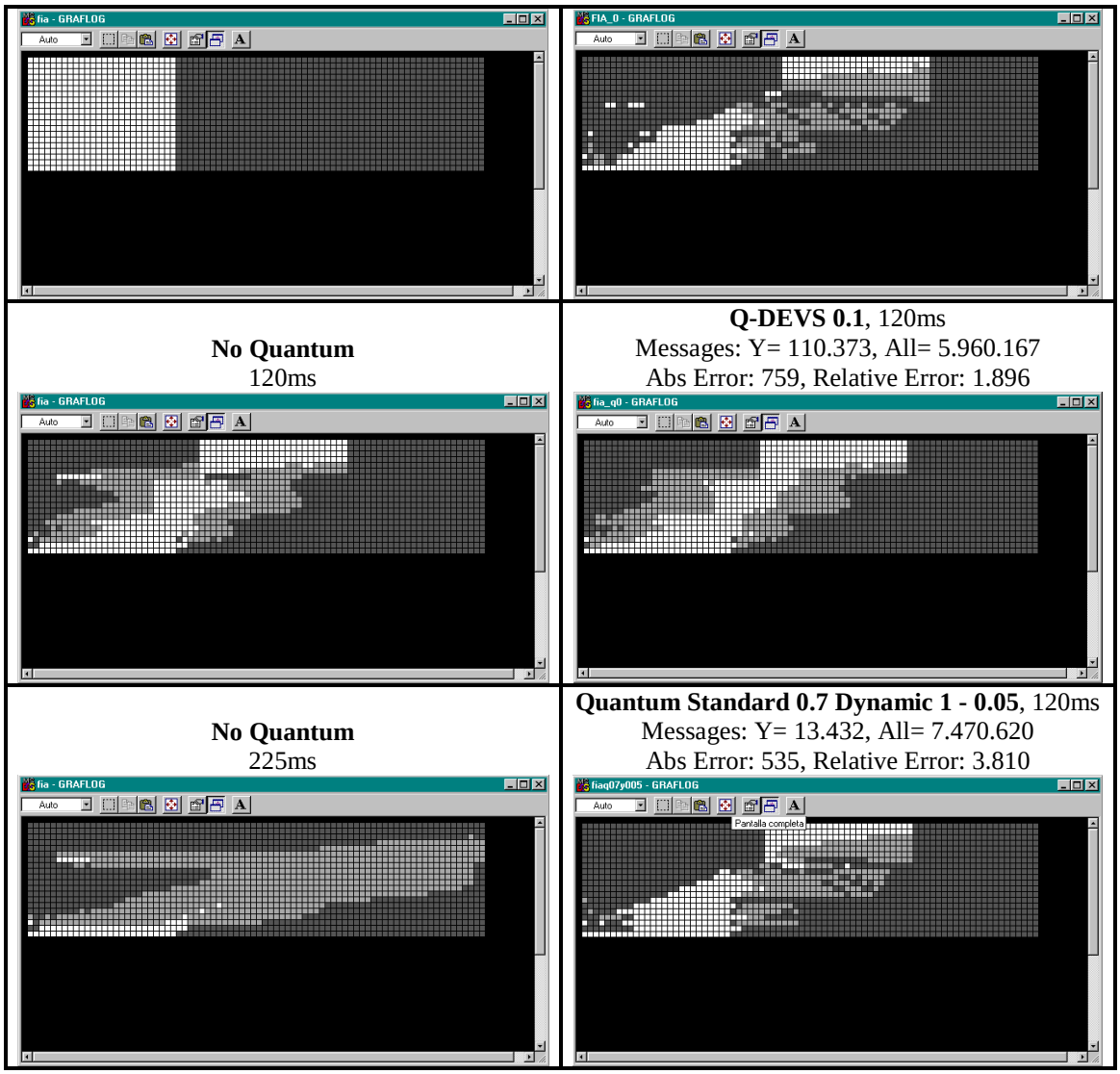


Figure 68. FIA Result Simulation Examples

Error analysis

With this model a different error behavior can be observed. E.g., with quantum Hysteresis dynamic strategy 1 ratio 0.5, at quantum value 0.1 the absolute error is 7337 and with the same quantum type with value 0.7 the error is 567. With a bigger quantum value we've got a lower error. This depends on the values of the simulation, which varies from 0 to 0.8 and with the translation function of FIA, with bigger quantum values, intermediate translations are avoided resulting on a lower error because of values compensation when no translating intermediate values.

Absolute Error analysis

On the next two figures (figure 69 & 70) we can see the absolute error of the standard and hysteresis quantums combined with the dynamic strategies and the Q-DEVS mechanism on the second figure.

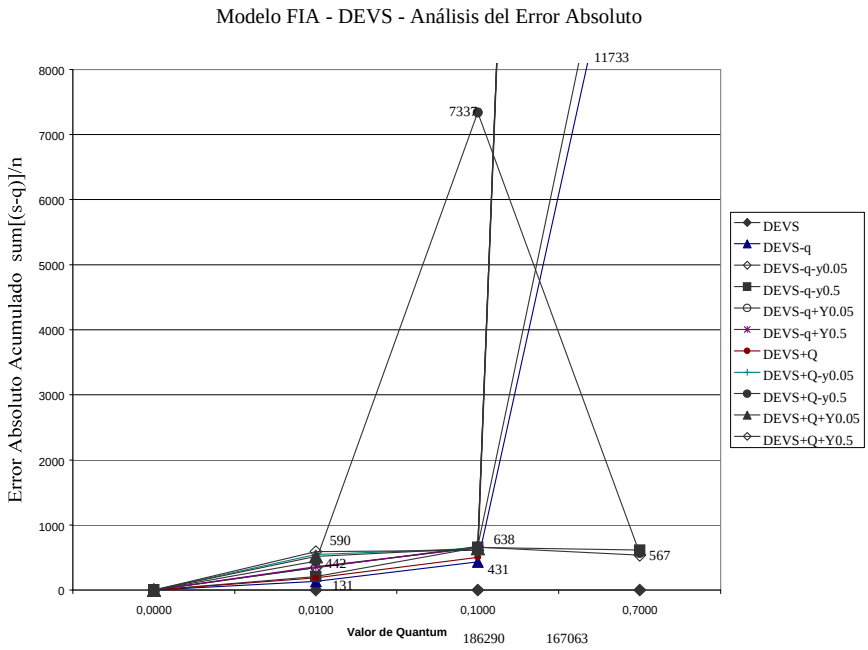


Figure 69. Cell-DEVS FIA model Absolute Error

On figure 69, we can see the higher error generated with the hysteresis dynamic quantum 0.1 compared with same strategy but with quantum value 0.7. Also with other techniques we have bigger errors also, like dynamic ratios 0.05 and 0.5 with quantum value 0.1

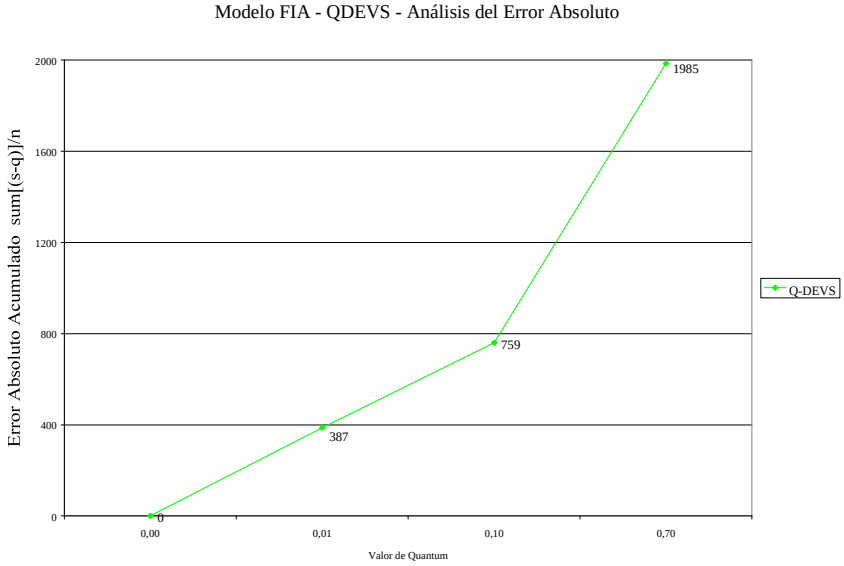


Figure 70. Q-DEVS FIA model Absolute Error

Figure 70 show the absolute error of the Q-DEVS mechanism, which has a better result with smallest quantums (like quantum value 0.01 were the error is 367).

Relative Error analysis

Figures 71 and 72 illustrates the relative error. As we can see, the behavior is quite similar to the absolute error, except for Q-DEVS technique (figure 72) were the difference of the relative error with quantum value 0.1 and 0.7 is not so different than the absolute error for same technique and quantum values.

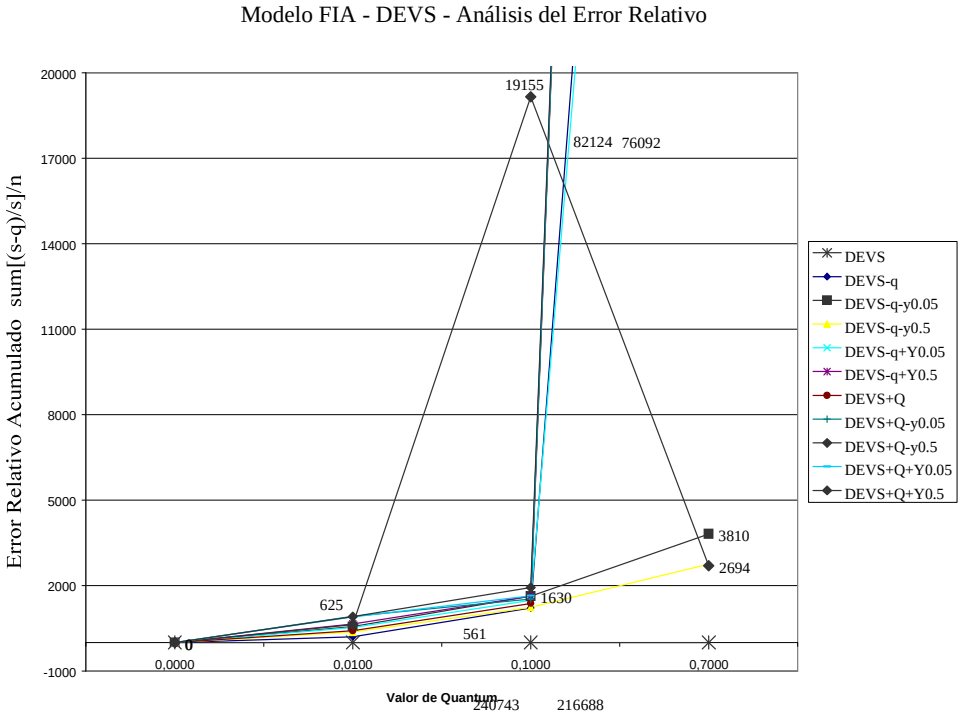


Figure 71. Cell-DEVS FIA model Relative Error

On next figure 72 we can see the relative error line of Q-DEVS with the difference mentioned before with respect to absolute error.

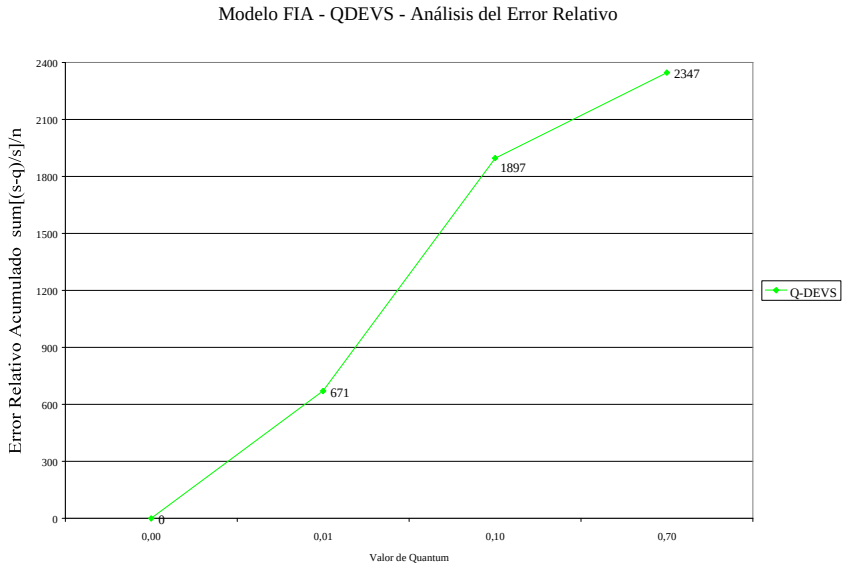


Figure 72. Q-DEVS FIA model Relative Error

Messages analysis

Quantum hysteresis dynamic strategy 1 ratio 0.5 with quantum value 0.1 it has the smaller number of messages reducing it from 7.000.000 to 4.000.000 messages while incurring on the biggest error.
This difference can be seen on the total message analysis, not on the output message analysis, where messages are always reduced when bigger the quantum is. On the total message analysis, with some smaller quantums, we have less messages than with bigger quantums, because of internal and external messages generated also when no output message is generated (as mentioned before, when no output message is generated, an external message is automatically generated to keep the simulation alive and update the local values).

Total Messages analysis

Next figure 73 show the total number of messages generated with the standard and hysteresis quantum (combined also with dynamic strategies). In this model we can see the relation of the number of messages with the incurred error.

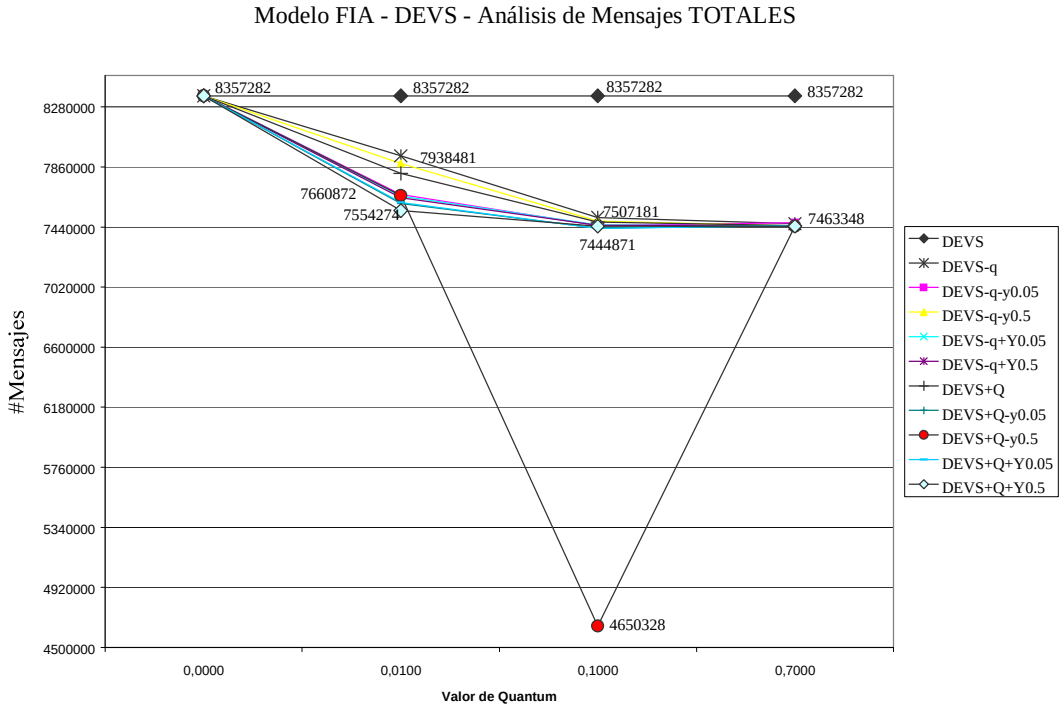


Figure 73. Cell-DEVS FIA model Total Messages

On this figure (figure 73) the lowest number of messages was obtained with the quantum hysteresis 0.1 dynamic strategy 1 ratio 0.5 On the error figures for this quantum type (figure 69 and 71) we can see that the error is the biggest, which is consistent with the lower number of messages obtained here.

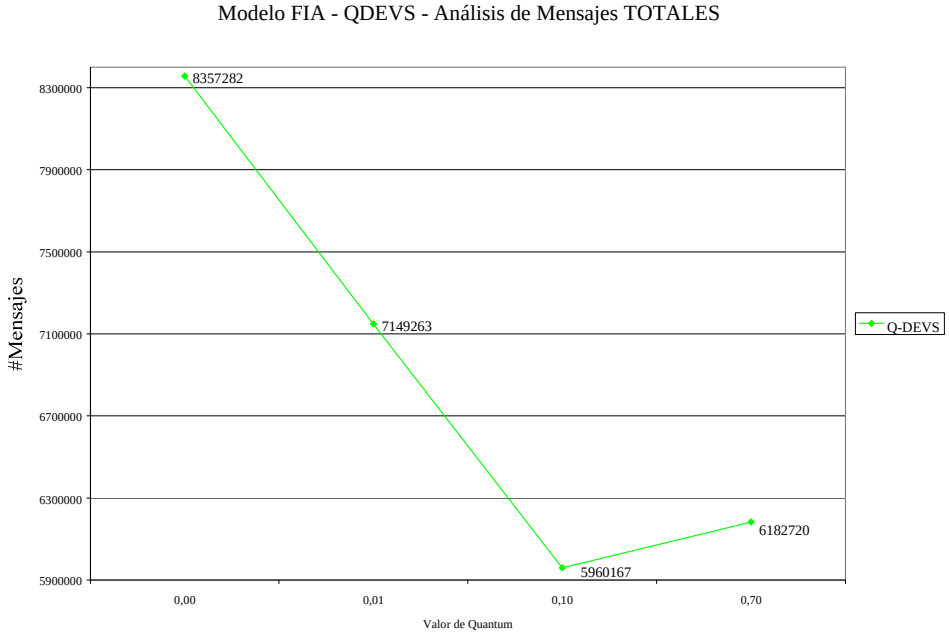


Figure 74. Q-DEVS FIA model Total Messages

This figures show the number of messages for Q-DEVS mechanism where we can see smaller number of messages with quantum 0.1 and not with quantum 0.7 were we have also a bigger error (as showed on figure 70 and 72). All in all, with Q-DEVS 0.7 we have more messages and higher error.

Output Messages analysis

Like other models, the output messages are reduced in a higher percentage than total messages.
On figure 75 we can see that the output messages are about 970.000 without quantum and are reduced up to 15.700 with quantum value 0.7. There are variations across the different types and strategies but reductions differences in output messages are not very important with 0.7 quantum value. With smaller quantum values, different techniques has more differences, for instance, with quantum 0.01 standard (no dynamic strategies) we have 584.000 output messages and with the same quantum value but quantum hysteresis with dynamic strategy 2 ratio 0.5 we have 204.000 messages (65% less messages).

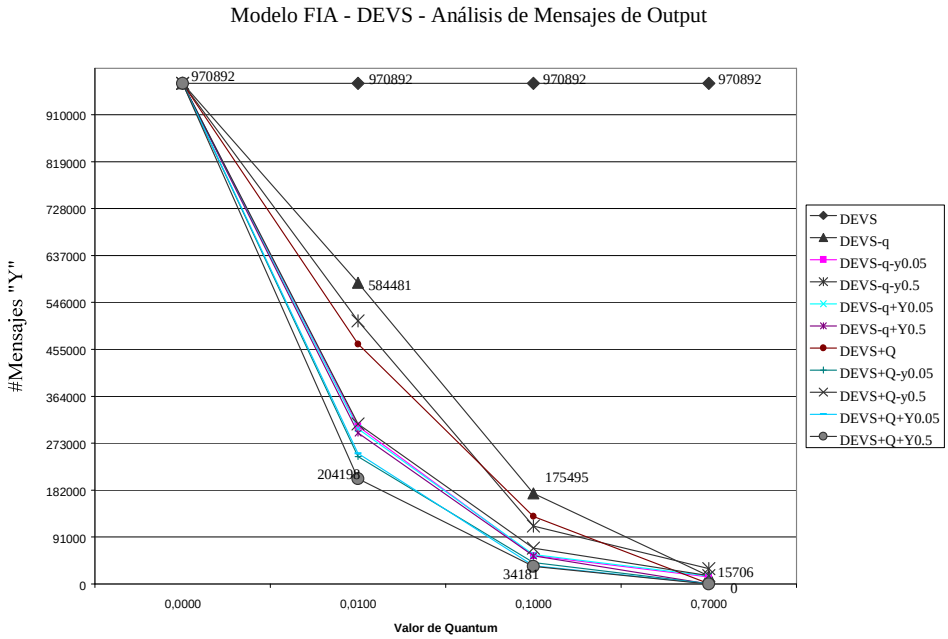


Figure 75. Cell-DEVS FIA model Output Messages

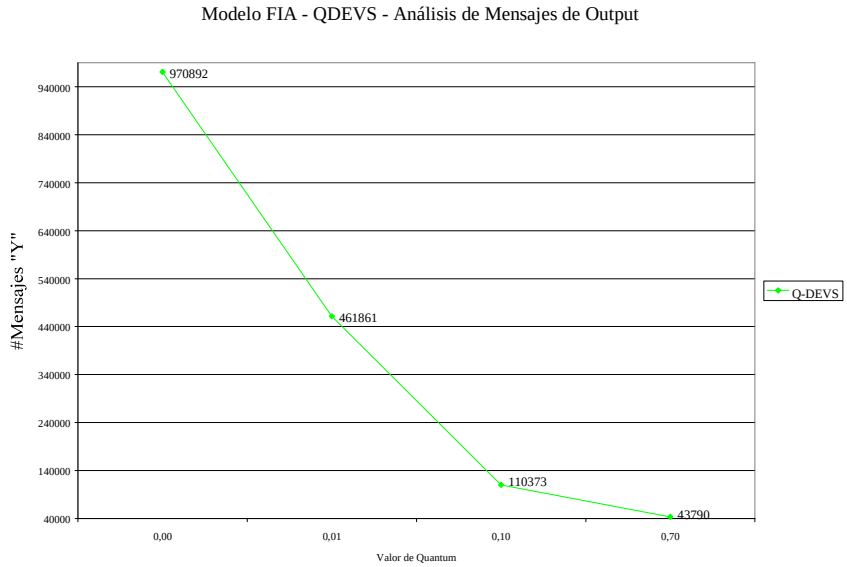


Figure 76. Q-DEVS FIA model Output Messages

This figure (figure 76) show the number of output messages for Q-DEVS mechanism. As we can see here we have an average of the messages generated with other techniques. For example, with Q-DEVS quantum 0.01 we have 461.000 output messages and with the standard quantums we have a minimum of 204.000 and maximum 584.000.

Simulation time analysis

Next two figures (figure 77 and 78) shows the simulation times for the different techniques.. The maximum time is about 22 minutes (1307 seconds) and the best time is about 19 minutes (1194 seconds) for the standard and hysteresis quantums and about 16 minutes (960 seconds) for Q-DEVS mechanism. The best saving is about 25%.

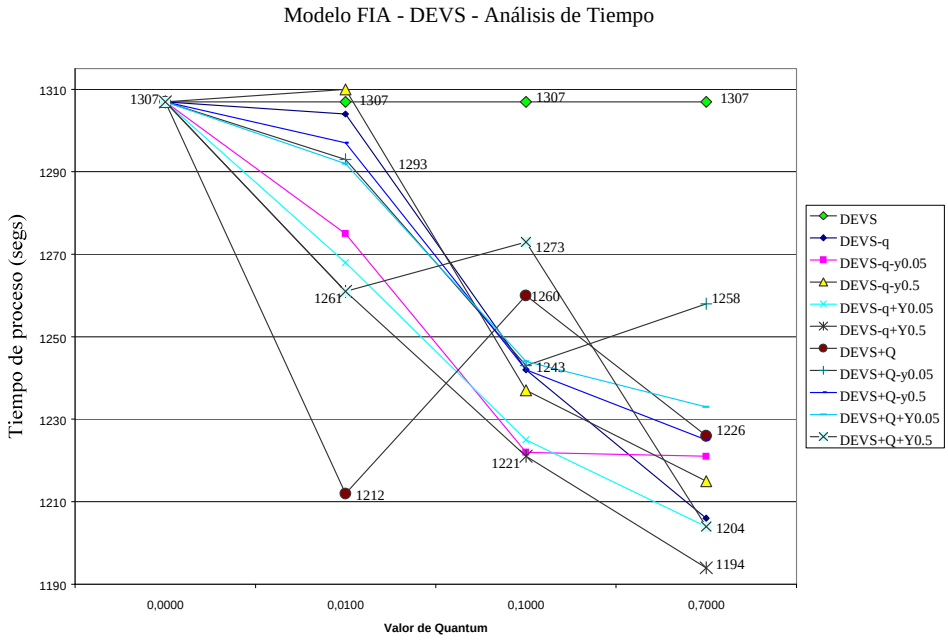


Figure 77. Cell-DEVS FIA model Processing Time

On next Figure 78 we have the total simulation time for Q-DEVS mechanism, where we can see that without quantum we have a very similar process time (1288 seconds for Q-DEVS vs. 1307 on the standard model) but with quantum we have the best time (25% less messages).

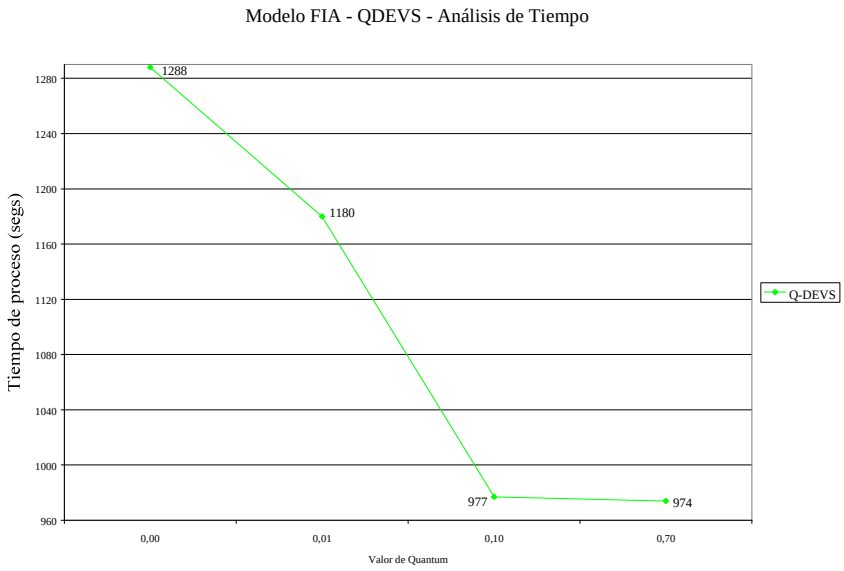


Figure 78. Q-DEVS FIA model Processing Time

8 Model Simulation Conclusions

Model, quantum type and quantum value best performance

The performance is defined as the lower absolute and relative error, lower number of output messages and lower processing time. The following sections shows the combinations of the most representative results for each model in terms of performance.

Heart Model

Table 3 shows a selection of the most representative results, in terms of performance (with the performance connotation defined before) for the Heart Model. The shadow lines shows the top 2 combinations with the best results for this model.

HEART MODEL	Total Messages	Output Messages	Absolute Error	Relative Error	Simulation time
NO Quantum	2.010.000	390.000	0	0	188”
Standard / Hysteresis 1.0 Dynamic Strategy 1 Radio 0.9	1.648.000	25.000	777	28	168”
Standard / Hysteresis 1.0 Dynamic Strategy 2 Radio 0.9	1.632.604	10.150	15.516	616	167”
Standard / Hysteresis 1.0 Dynamic Strategy 1 Radio 0.5	1.631.429	8.975	3.828	157	167”
Standard 1.0	1.628.529	6.075	7.674	337	164”
Q-DEVS 1.0	31.400	6.075	7.674	337	75”
GDEVS	10.761	2.076	1.388.980	31.913	3”
GDEVS + Standard / Hysteresis 20.0 Dynamic Strategy 1 Radio 0.9	8.836	150	1.026.750	29.726	2”

Table 3. Heart model Results Selection

Watershed Model

Table 4 shows a selection of the most representative results for the Watershed Model with the same performance definition of the Heart model. Here top 3 best combinations are shadowed because more combinations (more than the Heart model) with better results were obtained.

WATERSHED MODEL	Total Messages	Output Messages	Absolute Error	Relative Error	Simulation time
NO Quantum	1.347.300	268.800	0	0	129”
Standard / Hysteresis 1.0 Dynamic Strategy 1 Radio 0.9	1.092.398	13.800	27	0,29	111”
Standard / Hysteresis 1.0 Dynamic Strategy 2 Radio 0.9	1.079.243	743	341	3,8	114”
Standard / Hysteresis 3.5 Dynamic Strategy 2 Radio 0.9	1.078.741	241	8.905	98,3	113”

WATERSHED MODEL	Total Messages	Output Messages	Absolute Error	Relative Error	Simulation time
Standard / Hysteresis 1.0	1.079.688	1.188	259	2,8	113”
GDEVS	66.390	11.610	513	5,8	6”
Q-DEVS 1.0	4.956	536	559	4	8”
Q-DEVS 0.05	245.967	26.975	363	6,2	100”

Table 4. Heart model Results Selection

FIA Model

Table 5 has the best selected results for the FIA Model. Top 3 best combinations were shadowed (yellow). As we can see, with Q-DEVS 0.7 we have one of the best message reduction with ones of the lower errors for the different quantum combinations.

FIA MODEL	Total Messages	Output Messages	Absolute Error	Relative Error	Simulation time
NO Quantum	8.357.282	970.892	0	0	1.307”
Standard 0.1	7.507.181	175.495	431	1.212	1.242”
Hysteresis 0.1	7.475.633	131.039	503	1.371	1.260”
Standard 0.1 Dynamic Strategy 1 Radio 0.5	7.451.273	112.366	657	1.630	1.237”
Hysteresis 0.1 Dynamic Strategy 1 Radio 0.5	4.650.328	41.511	7.337	19.155	1.243”
Q-DEVS 0.1	5.960.167	110.373	759	1.897	977”
Hysteresis 0.7	7.450.086	416	167.063	216.688	1.226”
Q-DEVS 0.7	6.182.720	43.760	1.984	2.346	974”

Table 5. Heart model Results Selection

As we can see, the relative error is bigger than the absolute error. This is because on this model the values are under 1, which implies a relative error bigger than absolute. On the other two models (Heart and Watershed) values are bigger than 1, that’s the reason because relative error was smaller than absolute on that models.
Here Quantum Hysteresis dynamic strategy 1 ratio 0.5 has reduced significantly the number of messages while incurred on a bigger error, compared with the same value for quantum standard. Q-DEVS 0.7 has lower error than Hysteresis 0.7 but some more messages.

9 General Conclusions and problems to solve

The different quantums techniques were analyzed on this work and different models were used to make this analysis. Cell-DEVS formalism was analyzed and models with transport and inertial delay as well.
Parallel DEVS formalism and Cell-DEVS parallel executions with quantums were not analyzed on this work. The modifications made to CD++ were implemented on a version which has no parallel formalism implemented and needs to be implemented and tested on the parallel version.
Next sections shows conclusions and the different problems to solve and the impact of them on the quantization.

Problems to solve using Q-DEVS

When using Q-DEVS, a problem to solve is that Q-DEVS works with a function that is the inverse of the local update function and this inverse function returns the delay until next quantum region is achieved (and returns also the cell value for that time), which means that no activity will be made until the programmed time is achieved. The problem is when a cell needs its neighbors values to determine its own value, because the function is calculating the value for a future time (the time when the value will change its quantum region) and at the real time (when calculating the future value) neighbors values for each time are not available, because the neighbors has the values of the real time, not the values of the intermediate and future times that the analyzed cell is calculating.

For this work, to analyze the behavior of each technique, the models were transformed in order to minimize the neighbor dependency. For example, on the Heart model, every cell is updating individually its value without looking if the neighbors has positive or negative values (which is done on the real model, because a cell starts its action potential when its neighbors has positive values).
Another issue to consider is that if a function does not change its value (it returns always the same value, until ∞ Time), the system will loop forever waiting that the cell changes its value (its region) with a delay of ∞ as well. To avoid this problem and make possible the analysis and comparison of the different techniques, an additional condition was written in the inverse functions to avoid this problem: when no changes are made for a cell on the computation of the inverse function, the next value is set with the current value and the delay is set with the default delay. No changes is desired after n-steps without changes where n was set to 1 for this work.

Problems to solve using GDEVS

Like on Q-DEVS, the problem is to create a polynomial approximation when cells has different behavior. A possible solution can be to define a specific polynomial approximation for each cell. It can be very difficult and costly to obtain a valid approximation if cells has different behavior depending on its neighbors. On the FIA model for example, some cells starts with the 0 value and others with the 0.8 value. Cells changes its values due to a diffusion rule and to a transport rule, which means that cells from the left side will get value of the right cells (transport). Also at different times, cells will change its value with an average of their neighbors values (diffusion). The problem is with the different behavior of the cells, because cells representing the sample of the FIA model, will have a different behavior of the cells that are not close to the sample. If we see the generated values, to get an acceptable approximation, we need a polynomial approximation for each cell. For example, cell 0 0 starts with value 0 on time 0 and it changes to value 0.8 at time t+1 and that is the last change for that cell. Cells closer to the middle has values 0.5 and they will change its value abruptly to 0.8 when transport is made, but cells far of the sample will not change its values. The next figure shows the first 100 values for four different cells of the FIA model and the need of use different polynomial functions to approximate them. On Figure 79 (for cell 41 3) we can see the effect of the transport action, which alternate the cell value. That is an example of the very different behavior of cell 0 0.

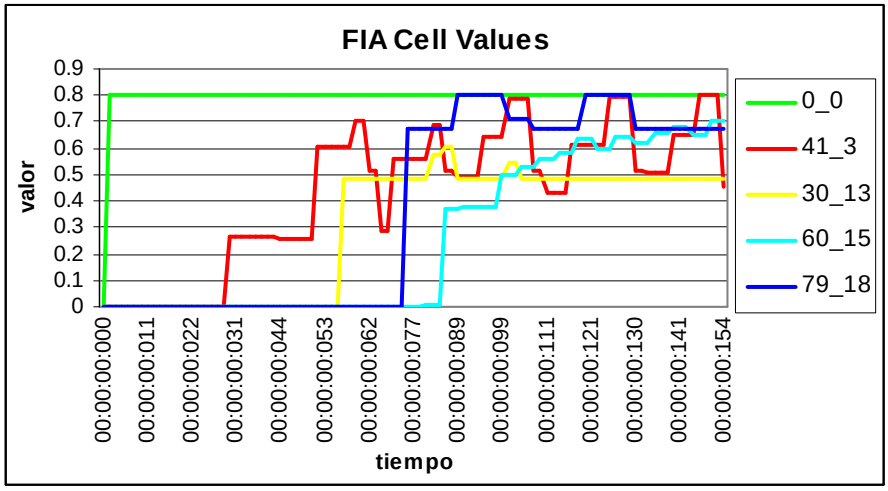


Figure 79. FIA model cell 41:3 variation

It depends on the model how convenient can be design a GDEVS representation. For models with abrupt changes and non uniform behavior, this technique can be hard to use, like on the FIA model.

N-arguments problem

On the CD++ version used for this work, the functions used on the models, can receive maximum two arguments. This will be enhanced on other works over CD++. To develop this work, two alternatives were used in order to bypass this limitation and continue with the quantum analysis. One was using two composed arguments (packed arguments) for the calculation functions. This was used on the Heart

model, which has 5 arguments and most argument values are integers so some packing algorithm can be applied easily.
On all Q-DEVS implementations and on the Cell-DEVS Watershed and FIA models, it's not possible to pack the arguments easily, because there are more arguments and there are more than two real arguments, so it's not possible to pack them into two arguments, so a new alternative was developed in order to implement this models. This alternative implements two functions called setArg and getArg, where setArg sets the i^{th} argument for the indicated cell and getArg returns the i^{th} argument for the indicated cell. This functions works as follows:

To set an argument value: setArg(x*100000+ y*100+i, α)
To get an argument value: getArg(x*100000+ y*100+i)

Where:

x is the x position of the cell
y is the y position of the cell
i is the argument number
α is the value for the argument

The composition of the x and y values was made because this function was implemented in CD++ which has the two arguments restriction.
With this two functions, up to 99 arguments can be used for internal functions and both (setArg and getArg) can be used inside the functions and inside the definition model as well.

X messages with –c argument

When simulating models with small variance or when quantifying simulations, it can occur the simulation stops because non-external events are generated (simulation stops when no external events are or when the ending times is achieved).
When using quantum, is common that on some specific times no events are generated as a result of the quantization. One solution to this is to define an additional plane to keep the model alive (with this additional plane varying from 1 to 0 and from 0 to 1 for example, and of course, not quantifying this plane). The problem with this solution is that it disturbs the optimization because more messages are generated with this additional plane.
A new argument for CD++ was implemented in order to enhance this behavior and to allow quantization analysis without over heading the message generation with an additional plane.
This argument indicates CD++ that a discrete advance time is in use and when a cell has no external events for a time t and is not influenced by any event, an external event will be generated with the indicated delay with the same effect of the additional plane but without over heading the messages generated and not disturbing the performance and quality analysis of the model and quantization. This argument it automatically generates "X" messages (external messages) for the cell without events.
This is the reason because when comparing non quantized and quantized simulations, the total number of messages are not reduced so significantly as the output messages, which are reduced as much as desired with the quantum value.
Of course, when using Q-DEVS mechanism, this is different because the quantum is not on the CD++ simulation, it is on the implemented function for the model and the next scheduled time for the cell manages the simulation (so this problem is not present). That's the reason because Q-DEVS mechanism reduces the total number of messages as much as the output messages, because the simulation is held until the next scheduled time is achieved and in between no internal or external events are generated.

10 Appendix – Tools & Functions

The output files and results are too big to be managed with an standard editor. Several tools where developed to analyze the results of the quantized simulations.

10.1 How to create complex functions in CD++

To make a new function available to use on the model file (.ma) its necessary to do the following steps.

1) Write the function in C++.
2) Copy this function into the Real Functions source file (realfunc.cpp). Replace the type of the arguments by the CD++ types. For example, "double" type have to be replaced by "Real" type.
   Example:

```
Real MyFunc(const Real &r1, const Real &r2)
{
 #include <math.h>
 #include <stdio.h>

 double var;
 var=r1.value()*r2.value() – 450.5;

 return Real(var);
}
```

3) Add the prototype of the function in the Real Functions header file (realfunc.h).
   Example:
```
Real MyFunc( const Real &r1, const Real &r2 );
```

4) Overload the operator( ) of the Z class for the structure of the function with the correct kind (unary, binary, etc.) on the same header file (realfunc.h).
   Example:
```
template <class T, class Z>
struct r_MyFunc : public binary_function< T, T, Z>
{
        Z operator()(const T& t1, const T& t2) const
        {
                if (EvalDebug().Active())
                        EvalDebug().Stream() << " (myfunc) ";

                return MyFunc(t1,t2);
        }

        string type(){ return "MYFUNC";}
};
```

5) Create the type of the function r_MyFunc on the same header file (realfunc.h).
   Example:
```
typedef r_MyFunc< Real, Real >  REAL_MYFUNC;
```

6) Define the respective operator for the class to manage the type value for the new function on the synnode.h header file.
   Example:
```
typedef BinaryOpNode< REAL_MYFUNC, RealType, RealType > FuncMyFunc ;
```

7) Add the name and type of the new function on the dictionary of parser method (parser.cpp source file).
   Example:
```
        dict[ "myfunc" ] = ValuePair( BINARY_FUNC, new FuncMyFunc() ) ;
```

8) Compile the project.

After this, the new function is available to use from the model.
Example:
```
rule : { Myfunc(cellpos(0)*1000+cellpos(1)*10+ if( (-1,0) > 0, 1.0, 0.0) } 5 {t}
```

10.2 Error calculation

The program ERRORQ accept a DRAWLOG output and generates a new output (on standard output – must be redirected to a file--) with six columns. The DRAWLOG output must be generated with the lines and times titles (no arguments –f or –e). Other way, comparison is not possible, because a different offset

on each file can occur. Only argument –e can be useful because the timeline is needed to compare the outputs.
The generated columns are:

0) Counter of times simulations.
1) Time of the block-simulation in comparison
2) Relative error: (1-s/q)/n = Error introduced on the indicated time. Not summarized (only the error generated on this time).
3) Accumulated relative error: sum[(1-s/q)]/n= The same as 2 but accumulated until current time.
4) Absolute error: (s-q)/n= The same as 2 but with a different formula. Not summarized (only the error generated on this time).
5) Accumulated Absolute error: sum[s-q]/n= The same as 4 but accumulated until current time.

Kind of calculations
Columns showed
Column 0

Line Counter

Column 1

Simulation Time

Column 2

Relative error: (1-s/q) / n = $\sum$ (1-s_i/q_i)/n (for 0<=i <=n = number of cells)
s_i = Value of cell i on the output without quantum.
q_i = Value of cell i on the output with quantum.
n = number of cells
This value is set to 0 after each block-time.

Column 3

(is the same as 2 but accumulated -- Accumulated Relative error--)
sum[(1-s/q)]/n = $\sum$ [$\sum$ (1-s_i/q_i)]$_j$ /n (for 0<=i <=n = number of cells, 0<=j<=current time)
s_i = Value of cell i on the output without quantum.
q_i = Value of cell i on the output with quantum.
n = number of cells

Column 4

Absolute error: (s-q) / n = $\sum$ (s_i-q_i)/n (for 0<=i <=n = number of cells)
s_i = Value of cell i on the output without quantum.
q_i = Value of cell i on the output with quantum.
n = number of cells
This value is set to 0 after each block-time.

Column 5

(is the same as 4 but accumulated --Accumulated absolute error --)
sum[(s-q)]/n = $\sum$ [$\sum$ (s_i-q_i)]$_j$ /n (for 0<=i <=n = number of cells, 0<=j<=current time)
s_i = Value of cell i on the output without quantum.
q_i = Value of cell i on the output with quantum.
n = number of cells

When the error is accumulated, means the error accumulated until time indicated in column 1.

Program arguments

The program ERRORQ receives two arguments:

1) The name of the original draw output file (original means without quantum)
2) The name of the draw output file obtained with quantum

Both files must be generated with DrawLog without the option –f, because the time is needed to synchronize the files and calculate the error.

Example

errorq output1.drw outputqq.drw

This will show you:

```
Archivo original: sinq.drw quantificado:conq.drw
Dimension detectada del modelo: 5 x 5
  Descripcion de Columnas
    0) Contador de bloques comparados
```

```
      1) Tiempo de simulaicon del bloque en comparacion
      2) (1-s/q)/n= Error '/' introducido en el bloque time. Sin
acumular.
      3) sum[(1-s/q)]/n= Error '/' acumulado hasta time.
      4) (s-q)/n= Error '-' introducido en el bloque time. Sin
acumular.
      5) sum[s-q]/n= Error '-' acumulado hasta el bloque time.
  0,1,2,3,4,5
  t,time,(1-s/q)/25,sum[(1-s/q)]/25,(s-q)/25,sum[(s-q)]/25
  0,00:00:00:000,0,0,0,0
  1,00:00:00:230,0,0,0,0
  2,00:00:00:325,0,0,0,0
  3,00:00:00:355,0.00289157,0.00289157,0.24,0.24
  4,00:00:00:595,0.0446277,0.0475192,0.459642,0.699642
  5,00:00:00:680,0.0580914,0.105611,0.868588,1.56823
  6,00:00:00:710,0.056741,0.162352,0.981349,2.54958
  7,00:00:00:945,0.115058,0.27741,3.8915,6.44108
  8,00:00:00:995,0.0714215,0.348831,1.33647,7.77755
  9,00:00:01:030,0.0944933,0.443325,1.65318,9.43073
  10,00:00:01:060,0.105529,0.548853,2.00511,11.4358
  11,00:00:01:095,0.161452,0.710305,2.69788,14.1337
```

if you do (for example, in MSWindows):

errorq output1.drw outputqq.drw > error.csv

This will generate a comma separated value.

10.3 Message accounting

The program CONTART accept a simu LOG and gives you the number of messages used on that simulation.

Program arguments

The program CONTART receives one or two arguments (depending of the use)

1) The name of the SIMU LOG file
2) <optional> The Time until you want to count messages.
3) <optional> “d” to get the number of messages on each time.

Example1

contart output.log

This will show:

```
  Archivo a contar mensajes: output.log
  Cantidad de mensajes en output.log hasta 00:02:00:000(EOF) -> 264878
  #*=72290
  #X=48000
  #Y=24290
  #D=120294
  #I=4
```

This means that in the log output.log, until EOF (because the title “hasta 00:02:00:000(EOF) means that end of file was reached) there are a total of 264878 messages and this is the detail:
72290 messages of type “*”
48000 messages of type “X”
24290 messages of type “Y”
120294 messages of type “D”
4 messages of type “I”

If you use the second argument, the program will count messages until the indicated time is reached or EOF (the first that occurs).

Example2

With the same log as Example1, we can do:

contart output.log 00:01:10:250

And this will show:

```
  Archivo a contar mensajes: output.log
  Hasta: 00:01:10:250
  Cantidad de mensajes en output.log hasta 00:01:10:250 -> 155011
  #*=42301
  #X=28100
  #Y=14201
```

```
#D=70405
#I=4
```

This means that in the log output.log, until 00:01:10:250 simulation time (because the second parameter is in use "hasta 00:01:10:250") there are a total of 155011 messages. If you use, for this example,

contart output.log 00:03:00:000

You will get the same results as in Example1, because EOF will be reached before the 3 minutes indicated on the 2nd parameter, and you will see the EOF title as in example 1.

This is useful when you logs comparison of simulations ended at different simulation-times is needed, which is common when quantifying.

Example3

With the same log as Example1, we can do:

contart output.log d

This will show:

```
Archivo a contar mensajes: output.log
Time, Tot, *, X, Y, D, I
00:00:00:000,1,0,0,0,0,1
00:00:00:005,105,1,0,0,52,52
00:00:00:010,287,29,50,26,130,52
00:00:00:015,469,57,100,52,208,52
00:00:00:020,651,85,150,78,286,52
00:00:00:025,833,113,200,104,364,52
00:00:00:030,1015,141,250,130,442,52
00:00:00:035,1197,169,300,156,520,52
00:00:00:040,1379,197,350,182,598,52
00:00:00:045,1561,225,400,208,676,52
00:00:00:050,1743,253,450,234,754,52
... etc.

Cantidad de mensajes en output.log hasta 00:01:10:250 -> 155011
#*=672000
#X=1200000
#Y=624000
#D=1872052
#I=52
```

10.4 Stamp printer

Stamp is a program that shows system date and time. No arguments.

e.g.:

stamp

will produce

yyyy-mm-dd hh:mm:ss

with the current date and time in that format.

10.5 Cell Extractor

The program EXTRAER accept a DRAWLOG output and produces a new output (on standard output – must be redirected to a file--) with the values of the indicated cell.

Program arguments

The program EXTRAER receives three arguments:

1) The name of the draw output file
2) The cell x position
3) The cell y position

Example

extraer output1.drw 2 1

This will show you the time and values for the cell x, y of the output1.drw DRWLOG output.

extraer output1.drw 2 1 > x21.csv

This will generate a comma separated value.

10.6 Massive simulations

The program RUNSIMUS accept a file with a list of quantum values to consider for the simulations. This program will automatically run all the tools necessaries for the analysis of the quantum.

Programs/Tools used

0) CD++ Simulation.
1) DrawLog
2) ContarT (To get the number of messages from the simulation).
3) Extraer 0 0 (To get the values of the cell 0,0)
4) ErrorQ (Error calculation of the simulation compared with the simulation without quantum)
5) Stamp (To get the system time stamp at the beginning and at the end of the simu)

Program arguments

The program RunSimus receives between three and four arguments:

1) The path/file name of the ASCII file with all the quantum arguments type and values to be used.
2) Arguments to be used for the simulation
3) Arguments to be used for drawlog
4) Starting name for the output files to be generated
5) Indicator to process or not the initial simulation for the comparison base.

Examples

newrunsimus parquantums.txt "-mcorazon.ma -t00:01:18:000 -c00:00:00:005 -s -d0.0000001 " "-mcorazon.ma -ccorazon -w12 -p5 " cora

Example of a quantum values file

parquantums.txt content:

```
%sin quantum

%quantum standard
-q1.0
-q8.0
-q20.0
%quantum standard dinamico 1
-q1.0 -y0.05
-q1.0 -y0.5
-q1.0 -y0.9
-q8.0 -y0.05
-q8.0 -y0.5
-q8.0 -y0.9
-q20.0 -y0.05
-q20.0 -y0.5
-q20.0 -y0.9
%quantum standard dinamico 2
-q1.0 -Y0.05
-q1.0 -Y0.5
-q1.0 -Y0.9
-q8.0 -Y0.05
-q8.0 -Y0.5
-q8.0 -Y0.9
-q20.0 -Y0.05
-q20.0 -Y0.5
-q20.0 -Y0.9
%quantum histeresis
-Q1.0
-Q8.0
-Q20.0
%quantum histeresis dinamico 1
-Q1.0 -y0.05
-Q1.0 -y0.5
-Q1.0 -y0.9
-Q8.0 -y0.05
-Q8.0 -y0.5
-Q8.0 -y0.9
-Q20.0 -y0.05
-Q20.0 -y0.5
-Q20.0 -y0.9
%quantum histeresis dinamico 2
-Q1.0 -Y0.05
```

```
-Q1.0 -Y0.5
-Q1.0 -Y0.9
-Q8.0 -Y0.05
-Q8.0 -Y0.5
-Q8.0 -Y0.9
-Q20.0 -Y0.05
-Q20.0 -Y0.5
-Q20.0 -Y0.9
```

This example runs the simulations and tools for all the quantum types and values indicated in parquantums.txt file.

Outputs to obtain

The outputs files obtained in this example are of the form:

cora_XXXXX.txt for the standard output generated by the tools.
cora_XXXXX.log for the log file of the simulation. (this file is automatically deleted once the simulation and tools running ends)
cora_XXXXX.drw for the drawlog output file.
cora_XXXXX_#.txt for the message counter resume.
cora_XXXXX_x00.csv for the 0 0 cell value counter resume.
cora_XXXXX_e.csv error comparison of the simulation.

Where XXXXX is each of the quantum type and values parameters indicated on parquantums.txt file.
e.g.:

cora_-q1.0 -y0.05.txt is a valid output name.

Considerations

The –Q and –Y arguments are replaced by +Q and +Y respectively in order to keep uniqueness file names of the output files (because some O.S. has no case sensitive file naming)

10.7 Massive comparisons

The program CMPSIMUS receives a file with a list of quantum values used on the simulations and generates a unique file with all the compared values for each quantum type and value indicated.

Program arguments

The program RunSimus receives two arguments:

1) The path/file name of the ASCII file with all the quantum arguments type and values used during the simulation.
2) Starting name for the output file to be generated

Examples

newcmpsimus parquantums.txt comp.
This will generate a comma separated file as follows:

TipoQuantum	ValorQuantum	TipoDinamico	ValorDinamico	Par	Inicio	Fin	#MsgsTOT	#MsgsY	ErrAbs	ErrRel
q	1			'-q1.0	2003-07-09 18:19:49	2003-07-09 18:22:33	1628529	6075	7674.24	337.478
q	8			'-q8.0	2003-07-09 18:24:12	2003-07-09 18:26:55	1623579	1125	69355.2	3020.86
q	20			'-q20.0	2003-07-09 18:28:28	2003-07-09 18:31:10	1622929	475	196829	7879.13
q	1.0_	y	0.05	'-q1.0_-y0.05	2003-07-09 18:32:43	2003-07-09 18:35:27	1628829	6375	7164.11	308.143
q	1.0_	y	0.5	'-q1.0_-y0.5	2003-07-09 18:37:00	2003-07-09 18:39:44	1631429	8975	3828.7	157.925
q	1.0_	y	0.9	'-q1.0_-y0.9	2003-07-09 18:41:17	2003-07-09 18:44:02	1648079	25625	777.12	28.5052
q	1.0_+Y0.05			'-q1.0_+Y0.05	2003-07-09 19:11:15	2003-07-09 19:13:59	1628579	6125	8205.94	361.65
q	1.0_+Y0.5			'-q1.0_+Y0.5	2003-07-09 19:15:33	2003-07-09 19:18:17	1628629	6175	11627.6	518.544

10.8 Graphical Cell drawer

The program GRAFCELL accepts a DRAWLOG output and shows a graphic output on the screen with all the cells and the function graphic for each one on a Grid.

Program arguments

The program GRAFCELL receives between five and seven arguments (depending on the use)
(On the next description, x means the values for x edges and y the values for y edges of the plane graphic)

1) The name of the DRAWLOG output file.

2) The minimum value for x (most times is 0 –cero—because of the starting time of a simulation is cero).
3) The minimum value for y. This is the minimum value that a cell can reach on the simulation.
4) The maximum value for x. This is the maximum time of the simulation, converted to milliseconds, but however, not all simulation times are showed on the drawlog, so you will need to adjust this parameter with the correct scale of the grid.
5) The maximum value for y. This is the maximum value that a cell can reach on the simulation.
6) <Optional> -t With "-t" argument, GrafCell will show the current time of the simulation when drawing. The time will be converted to milliseconds and divided by the default cell delay, but however, not all times simulations are showed on drawlog, so a better adjustment will be necessary.
   If the drawlog file does not include the titles and times, a counter will be showed. WARNING: With this option, the drawing can be VERY SLOW, depending on the model size.
7) <Optional> cellx celly With "x y" argument, GrafCell will show only the cell "cellx, celly".

NOTE

Arguments 2, 3, 4 and 5 are only to adjust the scale of the grid. You can try different ones to have a nicer view.

Example

GrafCell output.drw 0 -16.6 35000 94.3

This will show the graphics on a new window screen. GrafCell will automatically detect the number of cells

GrafCell output.drw 0 -16.6 35000 94.3 3 0

This will show only the graphic of the cell 3 0.

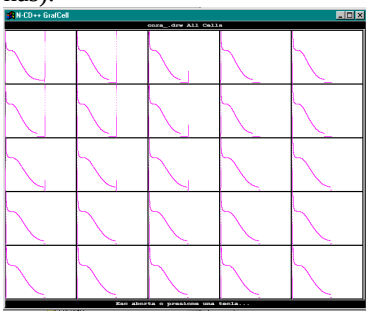
RESTRICTIONS

GrafCell will work properly only with drawlog outputs of one slide (parameter –f or –e needed).

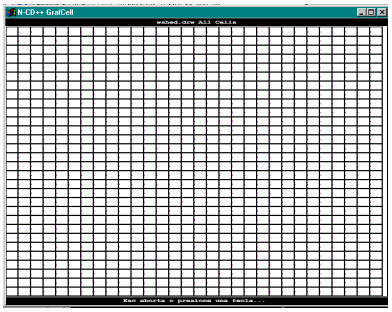
Suggestions

To graph an output of a simulation **with quantum**, is better to use a DRAWLOG output generated with –f or –e option, because –f or –e option shows outputs only for the necessary plane to show..

The execution will generate a graphical cell space as follows (it will show as many cells as the model has).



Heart GrafCell example with 5x5 cells



Watershed GrafCell example with 30x30 cells

11 Bibliography and References

[1] ZEIGLER, B. "Theory of modeling and simulation". Wiley, 1976.

[2] WAINER, G. "Discrete-events cellular models with explicit delays". Ph.D. Thesis, Université d'Aix-Marseille III. 1998.

[3] WAINER, G. "Improved cellular models with parallel Cell-DEVS". In Transactions of the SCS. June 2000.

[4] WAINER, G.; ZEIGLER, B. "Experimental results of Timed Cell-DEVS quantization". In Proceedings of AIS'2000. Tucson, Arizona. U.S.A. 2000.

[5] GOLDSCHLAGER, N.; GOLDMAN, M. "Principles of clinical electrocardiography". Appleton and Lange, Norwalk, CT. 1989.

[6] ALBERTS, B.; BRAY, D.; LEWIS, L.; RAFF, M.; ROBERTS, K.; WATSON, D. "Molecular Biology of the Cell" (First Edition) Garland Publishing, Inc., New York & London. 1983.

[7] WAINER, G.; Giambiasi, N.; CONTINOUS CELL-DEVS MODELS FOR COMPLEX SYSTEMS ANALYSIS

[8] HODGKIN, A.; HUXLEY, A. "A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve" Journal of Physiology 117: 500-544. 1952.
[9] SAXBERG, B.; COHEN, R. "Cellular Automata Models of Cardiac Conduction". In Theory of Heart. Springer-Verlag. 1991.
[10] ASCHENBRENNER, T. "The Hybrid Method: Simulation of the Electrical Activity of the Human Heart . International Journal of Bioelectromagnetism. Number 2, Volume 2, 2000.
[11] FENTON, F. "Numerical Simulations of Cardiac Dynamics. What can we learn from simple and complex models?". IEEE Computers in Cardiology, Vol. 27, 251-254. 2000.
[12] SAN MIGUEL, L.; WAINER, G. "Implementing the Hodgkin-Huxley model in CD++". Internal report, Universidad de Buenos Aires, Argentina. 2001.
[13] Bolduc J.S.; Vangheluwe H. "MAPPING ODES TO DEVS: ADAPTIVE QUANTIZATION"
[14] GIAMBIASI N.; GHOSH S. "Generalized Discrete Event Simulation of Dynamic Systems"
[15] MOON, Y.; ZEIGLE, B.; BALL, G.; GUERTIN, D., 1996. "DEVS representation of spatially distributed systems: validity, complexity reduction". IEEE Transactions on Systems, Man and Cybernetics. pp. 288-296.
[16] Ameghino J. "Aplicaciones de modelos celulares complejos usando N-CD++, FCEyN, UBA, 2000"
[17] Troccoli A. "Modificaciones a CD++ para simulación paralela y distribuida de modelos Cell-DEVS, FCEyN, UBA, 2001"
[18] GIAMBIASI, N. "Abstractions à événements Discrets de Systèmes Dynamiques". Journal Européen des Systèmes Automatisés (RAIRO APII, Automatique-Productique Informatique Industrielle - Hermes), Vol. 32, n° 3, pp. 275-311, 1998.
[19] NAAMANE, A.; GIAMBIASI, N.; BAKOP, D. "A new approach for discrete event simulations". IEE Electronics Letters, 34(16) pp. 1615-1616, 1998.
[20] KOFFMAN, E.; LEE, J.S.; ZEIGLER, B.P. "DEVS Representation of Differential Equation Systems: Review of Recent Advances".
[21] ZEIGLER, B.; KIM, T.; PRAEHOFER, H. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press. 2000.

12 Figures and Tables index