Krassimir Markov, Vitalii Velychko,
Lius Fernando de Mingo Lopez, Juan Casellanos
(editors)

New Trends
in
Information Technologies

I T H E A
SOFIA
2010

Krassimir Markov, Vitalii Velychko, Lius Fernando de Mingo Lopez, Juan Casellanos (ed.)

New Trends in Information Technologies

ITHEA®

Sofia, Bulgaria, 2010

ISBN 978-954-16-0044-9

First edition

Recommended for publication by The Scientific Concil of the Institute of Information Theories and Applications FOI ITHEA

This book maintains articles on actual problems of research and application of information technologies, especially the new approaches, models, algorithms and methods of membrane computing and transition P systems; decision support systems; discrete mathematics; problems of the interdisciplinary knowledge domain including informatics, computer science, control theory, and IT applications; information security; disaster risk assessment, based on heterogeneous information (from satellites and in-situ data, and modelling data); timely and reliable detection, estimation, and forecast of risk factors and, on this basis, on timely elimination of the causes of abnormal situations before failures and other undesirable consequences occur; models of mind, cognizers; computer virtual reality; virtual laboratories for computer-aided design; open social info-educational platforms; multimedia digital libraries and digital collections representing the European cultural and historical heritage; recognition of the similarities in architectures and power profiles of different types of arrays, adaptation of methods developed for one on others and component sharing when several arrays are embedded in the same system and mutually operated.

It is represented that book articles will be interesting for experts in the field of information technologies as well as for practical users.

General Sponsor: Consortium FOI Bulgaria (www.foibg.com).

Printed in Bulgaria

ISBN 978-954-16-0044-9

C\o Jusautor, Sofia, 2010

COLLISION DETECTION AND TREATMENT USING 2D RECONFIGURABLE HARDWARE

Alejandro Figueroa, Gustavo Méndez, Francisco J. Cisneros, Adriana Toni

Abstract: *The detection and treatment of collisions has been the subject of study for many years, periodically appear new techniques and algorithms to solve. it This article presents a hardware alternative to the detection of collisions between two or more surfaces in real time taking advantage of the parallelism offered by FPGAs, applying on a Spanish billiard of three balls simulator. FPGAs (Field-Programming Gate Array) provide a highly flexible environment for the programmer, since its cells can be reprogrammed and executed with great ease, which allows them to be used for an enormous range of applications.*

Keywords: *Collision detect, 2D Graphics*

ACM Classification Keywords: *I.6. Simulation and Modelling, I.3 Computer Graphics*

Introduction

The simplest definition of a collision detection between two objects (be they surfaces or volumes) is basically whether at a given moment, there is an intersection between the two or not. With this idea, we developed the overall study of collisions in an environment and its treatment [1].

For example, in three-dimensional model representation, the objects define their body - called mesh - through a large number of vertices that form polygons. These polygons are in contact with each other, forming the mesh. When there is a collision with an obstacle within the environment of the object, an application must determine where and how the intersection has occurred that caused the collision and take action.

Detection system and treatment of collisions is necessary for a large number of applications including entertainment systems, robotics, physical applications, biomedicine, military applications, etc. All these fields of research require such applications that solve their specific problems. Today, nearly all of these applications use collision detection software techniques developed in different languages, mostly software that needs one or more CPU to run [2] [3].

The FPGA is a semiconductor device, which contains a large number of logical blocks, called CLB, including a number of look-up tables (LUT) - where combinational logic is stored - "full adders" as basic hardware (adders), bistable , etc, whose interconnection and functionality is configurable by the programmer (which adds great flexibility to the developer). Each FPGA has more than one million CLBs, suggesting the enormous operational capability of these devices. In addition, FPGAs have special blocks for memory (both on chip and external) that act as a support for all the logic of the plate and, of course, can be freely programmed in accordance with the requirements of a particular system. [5] [6]

You can load a FPGA almost any program, provided that it does not exceed the capacity of the CLB or reports.

The FPGAs are reprogrammable, such as processors, which can be used to implement designs. These designs, once loaded onto the board, may be modified, and loaded again, as often as desired [4] [4].

The work presented here is a system of detection and treatment of two-dimensional collisions in a linear environment on a FPGA. To this end, this study is simulated by an application of the Spanish pool game (three balls).

The simulation uses the implicit parallelism of FPGAs to perform all calculations concurrently. Therefore, a formal language is needed to program, based on this inherent concurrency. VHDL has been chosen, one of the hardware description languages used in programming hardware on FPGAs.

VHDL conducts operations through assignment of signals in parallel through multithreading techniques that allow the programmer to modularize the tasks in different processes, leaving the machine that run concurrently. As in any software language, VHDL is very important for modularization, which is done by defining the entities that represent the behavior of the program.

Application and Features

The application is presented to the user via one VGA monitor, keyboard, and a speaker. With this, the user has at hand the complete management of the application.

The user controls through the keyboard's numeric pad for impact direction he wants to give the ball and simulate the impact of the cleat into the ball which will bounce with an initial velocity in the direction chosen.

The application is divided into the following modules:

- Sound Controller
- VGA Driver
- Keyboard Controller
- Main Application Module

(The modularization is detailed in the *Implementation* section)

The program has been done in several stages, in which features were added gradually. At first, he drew a polygon and speed is printed, regardless of collisions with the boundaries of the board. Thus, at first was a simple square movement, which eventually became a sphere. After obtaining the motion of the object, it was easier to detect collisions, within the limits of the board. The treatment of these collisions will be detailed below (paragraph *implementation)*. To the first white ball, direction was added from the keyboard, getting, finally, a more appropriate interface with which to control the start of the simulation. We subsequently added the friction between the pool table and the ball. It is also explained in the *logic of the System* section.

Once finished, the other two balls had to be added to the environment and make them collide. Indeed, this was the most difficult point with work, because internally, this resulted in several state machines working concurrently and synchronizing with each other. The detailed explanation about the WSF is detailed in the *logic of the System* section.

The last feature added to the application was a splash screen starting with the presentation of the game. It was through RAM at the beginning of the FPGA, reading the content stored in another memory and displays on screen *(Implementation* section in detail).

We have encountered a number of problems when implementing this program. In the experiments, we observed that the machine behaved satisfactorily when simulating shown and one or two balls simultaneously, but, however, sometimes failed to add the third ball. After many hours of experimentation, it was possible to isolate the problem and place it in concerning state machines. It concluded that the idea was flawed, but the parallelism of the main board, sometimes of a FSM signals propagated to another, with different clock signals (discussed further in the *logic of the system* section*)* were lost, resulting in erroneous behavior at times. It was decided to create a more consistent code that better optimize the capabilities of the FPGA without, at the same time, lost information caused by the lag between watches.

Logic System

The programming allows the user FPGAs have a logic high capacity for development, because of the huge amount of CLB that stores.

When you start to develop the work, you must take into account a number of conditions that are imposed by working on FPGAs. One concerns the graphical representation on VGA. Because the FPGA print in the monitor 2 pixel for each pixel in the Y axis (while in the X axis only paints one), we adapted the logical design of our system and values doubled when working on the Y axis. For this reason, the balls are not drawn as spheres, but as ellipses, whose Y component is twice the radius of the component X. The final representation on the screen is that of a regular field.

To try it this peculiarity, all signals were doubled to represent both the value on the axis X and the Y axis, because for the detection of collisions needed a complete record of the current positions of each object, and this requires real-time updating of the values.

Movement of the balls

The movement of the balls is done pixel by pixel (two pixels in the Y coordinate for each pixel in X). The speed of each ball is simulated using a parameterized clock, and modifying in real time the frequency of the clock, getting simulate the friction of the balls with the board, or wear after a collision. According to the value vector having the *direction* of the ball, the process updates the signals that indicate the position of the center of the ball.

Each ball has its own process of movement, which has a particular clock, for every ball should move independently. Each watch is parameterized by a vector, so that you can change the frequency at run time (note that increasing the value of that vector, decreases the clock frequency at which the process works, as seen in the snippet below).

```
PROCESS OF CLOCK SIGNAL FROM THE WHITE BALL
(SIMILAR TO THE PROCESS WILL WATCH THE OTHER BALL)
process (clock,reset)
begin
      if reset ='1' then
            WhiteClockCounter <="000000000000000000000000";
            WhiteClock<='0';
      elsif (clock'event and clock ='1') then
            WhiteClockCounter <= WhiteClockCounter +'1';
            if WhiteClockCounter>=WhiteParam then
                  WhiteClock <=not WhiteClock;
                  WhiteClockCounter <="000000000000000000000000";
            end if;
      end if;
end process
```

where *WhiteClock* is the clock signal the cue ball; *clock* is the clock of the FPGA,

WhiteClockCounter is the counter that counts the number of *clock* cycles that take *action.*

WhiteParam is the parameter that modifies the frequency. Compared with *WhiteClockCounter* and, if it has reached that number of cycles, it makes a transition from *WhiteClock.*

When the ball is at rest (state *S_ini)* the clock parameter is assigned a constant value. This value is small enough for the ball to go out with some initial velocity, but not as fast as for the player was not able to perceive.

In the process, in each clock cycle is increased the value of the vector, getting a longer clock cycle, giving the impression that the ball is stopping. To stop the ball, set a higher threshold for the vector, so if the state machine detects that it has exceeded that threshold, will transition to idle and the ball stops.

In the case of yellow and red balls, do not give an initial value vector that parameterizes the clock, because the ball will stand to suffer the impact of another ball. Upon the impact, the vector of the incised ball gets the value of the vector of ball that hit her. This generates a feeling that the ball is thrown to hit the same speed as the ball that coincided with it.

Finite state machines (FSM)

The Spanish pool consists of three balls. A white one the user hits, and two balls (red and yellow), which can only be hit by another ball. The behavior of the balls is implemented by three state machines. Each specifies the behavior of a ball. The state machines communicate with each other to report the state of the table, that is, the possible collisions between balls, walls, etc.

Different FSM differ little from each other, the behavior of the yellow ball is the same of the red and white, except that in the FSM of the cue ball must be controlled starting direction the user gives the ball. The three state machines work with the clock signal produced by the FPGA itself.

The implementation of each of the state machine in VHDL is by two concurrent processes. The first one is responsible for carrying out synchronously, according to the clock of the FPGA, the state transitions. The second process performs the operations defined for each state. It is responsible for monitoring the collisions, charge signals in the registers, etc.

PROCESS OF CHANGE OF STATUS:

```
process (clock,reset)
begin
      if (reset='1') then
            state <= Sini;
      elsif(clock'event and clock='1') then
            state<=nextState;
      end if;
end process;
```

PROCESSES. PSEUDOCODE

```
process (state,reset)
 begin
if (reset = '1') then if (reset = '1 ') then
      // Initialize SIGNALS
elsif (clock'event and clock = '1 ') then
      case state is
            when SIni=>
                  // INACTIVE STATUS
                  // TRANSITION TO S0 IF AND ONLY IF THE USER
                  Hit the ball
            when S0=>
                  // IF VECTOR PARAMETER> = THRESHOLD, TRANSITION
                   Sini
                  // IF NO, TRANSITION TO S1
            when S1=>
                  // Intermediate state. always, Transition to S2;
            when S2  =>
                  // Collision detection
                  // UPDATE VALUES OF DIRECTION OF THE BALL
                  // If there is another collision with ball, flag and active
            change your address
                  // TRANSITION TO S0
            when others =>
                   // TRANSITION TO S0.
            end case;
      end if;
end process;
```

The basic and most important property of the FPGAs is the natural parallelism offered to the developer. In the system, such parallelism is used to make balls move simultaneously.

Initially, the three state machines are *asleep,* that is, in its initial state *(S_ini).* When the users determines a direction using the keypad and confirm the release, this selection is loaded in the register which controls the direction of the cue ball and the state machine is *activated,* making a transition to the first state. The ball begins to move, as specified in paragraph *movement of the ball.* Collisions with both walls and other balls are treated in the state S2. If the ball detects a collision with the red or yellow ball, the signal corresponding to active mode *flag* is activated and received by the state machine of the impacted ball, charging in its address register orientation calculated from the direction of the ball incident (as explained in the *Implementation* section**,** subparagraph *collisions)* making a transition to its first state, ie the ball *wakes up* and starts to move.

The outline of the state machine is as follows:

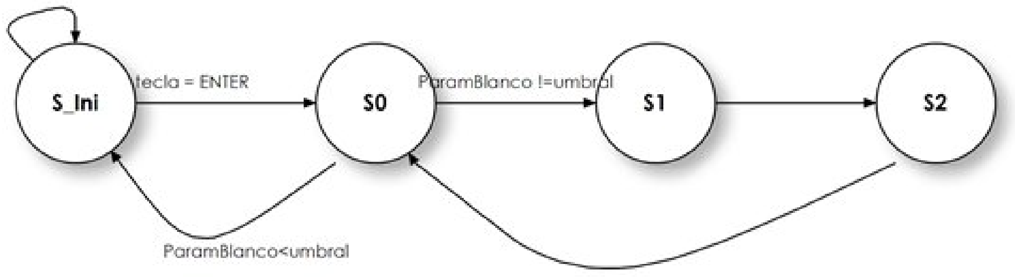

Fig 1. FSM white ball

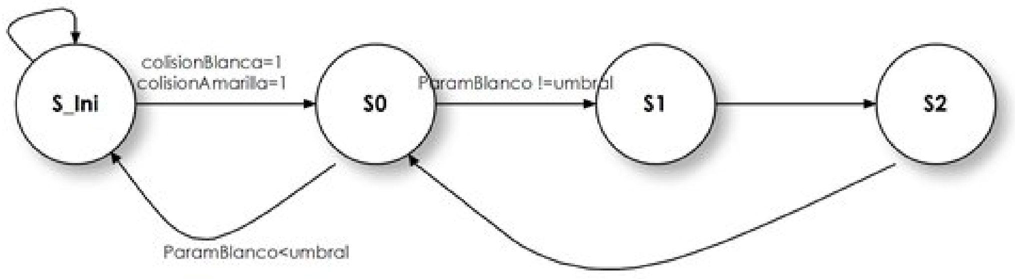

Fig 2. FSM yellow and white ball

It is seen that the three state machines are similar, have the same states and the same transitions, being the only difference between them how to be the first transition (from *S_ini* to *S0).*

The operation of state machines is as follows:

- ***S0:*** This state is the first state of movement of the ball, which moves during a cycle. This statement makes a check of the clock signal particular parameterization of the ball, to see if the threshold that will force the ball to stop has been reached. If, indeed, being reached, makes a transition to inactivity, S_Ini. If the check is counterfeit, it makes the transition to S1.
- ***S1:*** This state has the unique function of intermediate state. In this state, the ball does not move. It makes a transition to S2 in any case. This state is necessary, because thanks to the modification of the *Enable* movement, the FPGA plays S0, S1 and S2 as independent states and, overall, as a FSM.
- ***S2:*** The state that performs the audit of collisions. By order, checks if the ball hits the walls that delimit the table or whether, on the contrary, collide with another ball. For this test uses a sequence of *If - then-elsif-else* to check all cases. If a collision is detected, updates the new direction of the incident ball and

hit the ball activates the corresponding flag to notify the FSM of the ball and, finally, increases the vector that parameterizes the ball, to simulate the impact wear. After making all these calculations, it makes a transition back to S0.

The only state that differs from state machines is the initial state, which in the case of the white ball makes transitions to itself every cycle until the user hits the ball, when the FSM goes to S0. In the case of the other balls, this transition is only made when the impact *flags* are activated.

Implementation

Graphic representation

All the elements that represent the program are housed in the same process, which works with a special clock signal whose frequency is suitable for work on a VGA terminal.

The idea is to control, by means of two counters, the position of every pixel of your monitor. These counters *(hcnt* for the pixel count of the X axis, and *VCNT* for Y-axis) increase in each VGA clock cycle. With these counters, you can paint every pixel. Sequences are chained *if-then-else* to check the current values of these counters and assign a color to each region between them. Thus, the board is defined from the lines that delimit. The following snippet of code has been greatly simplified:

```
if ((hcnt<0 or hcnt>268) or (vcnt<18 or vcnt>301)) then
        rgb<="000000000"; --black
elsif ((hcnt>=0 and hcnt<269) and ((vcnt>17 and vcnt<33) or (vcnt>287 and vcnt<302))) then
        rgb<="011010000"; --brown
elsif ((vcnt>=24 and vcnt<295) and ((hcnt>=0 and hcnt<8) or (hcnt>261 and hcnt<269))) then
        rgb<="011010000"; --brown
else
        rgb<="000100000"; --green
end if;
```

With few lines of code, you can specify the color that should have the entire VGA monitor. The RGB signal is responsible for assigning a pixel color.

Lines are used *1, 2, 3, 4* of Figure 3 to define the edges of the table and use them to paint the table and collision handling.

For the initial screen using an image memory with a representative image, whose relationship pixel - memory location is performed similarly to the board. It also develops an VHDL entity that defines the behavior of a RAM. When the board starts, loads in memory an initial vector with the image you want to load and then the allocation is made of the corresponding pixel.

To choose the direction you use the number keys and the enter key to confirm and hit the ball in that direction. The representation of the direction is drawing the point of impact of bat on the ball (point *5* in the Fig 3).

The balls are created as a vertical ellipse twice the horizontal radius to compensate for both radio and create the effect of circumference.

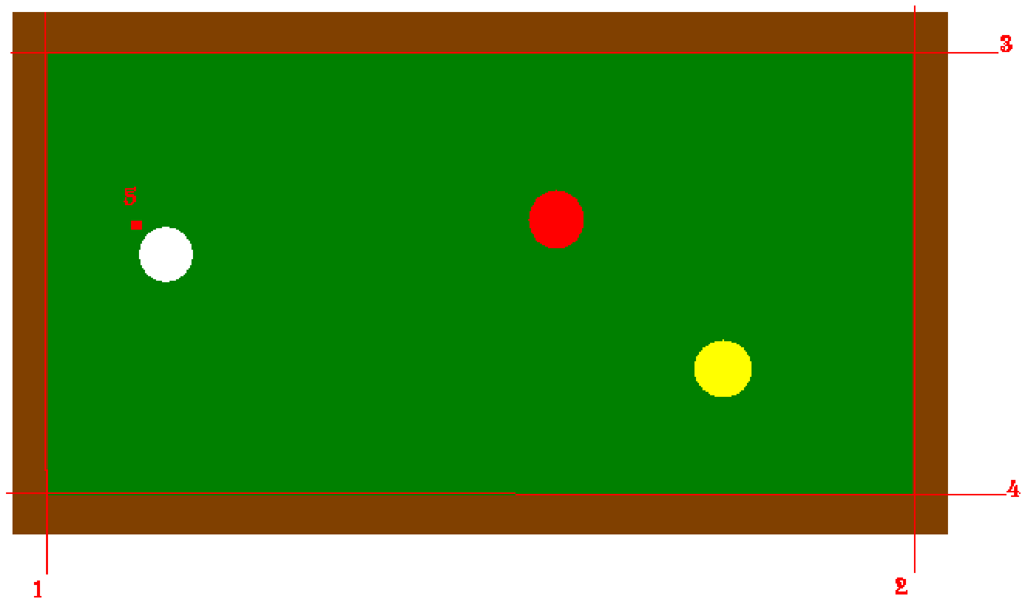

Fig 3. Board of game

Simulation

This section details how objects are defined internally affecting the simulation.

The main problem faced by a programmer to work with FPGAs is the arduous task of getting the FPGA code correctly interpreted the way you want. When, moreover, it is a system that modifies their values dynamically, we face the need to constantly store, in a series of signals, the updated values of the attributes and object in the system and define them for each conditional branch so that the system has a number of concrete data and no problems of interpretation.

The three balls are defined by two signals each, which store the position of its center in each component (X and Y) so that we know where the ball is located within the board. These two signals are essential in the calculation of collisions. Each clock cycle, during the simulation, these signals are updated depending on the direction of the ball. It is important to re-emphasize that the movement of the balls is done pixel by pixel, two by two in the case of Y.

The direction of each ball is also implemented with a signal, a vector of three bits, to specify the eight possible directions you can take a ball in the simulation. This vector is also of great importance for the calculation of collisions, it determines the output direction of the ball incident and incised.

Collisions

This section details how to detect, treat and resolve the application of the balls collisions with both other balls as the walls.

The motion simulation is pixel by pixel (explained in paragraph *movement of the balls* in the *Logic of the system* section). With this premise, we eliminate the problem of interpenetration.

The algorithmic scheme is based on the idea of covering each ball with a *Bounding-Box* that delimits. Thus, the balls are treated as if they were virtual square, significantly simplifying the logic needed to resolve these collisions.

As explained above, the application stores all the information of the objects in signals. It has a signal that indicates the radius of the balls. Is to create a Bounding-Box dimensions *2 * Radio.*

Collisions with fixed objects. Wall

These collisions are the simplest, as the walls no change after the collision.

As already indicated, is a comprehensive management of the values of the positions, and the limits of the board. Note that these limits are constantly required to paint the board.

Because the determination is carried out in the state machine, you can make that during that cycle the ball remains stopped until the collision is resolved. The user does not appreciate this stop. During that cycle (coinciding with the state *S2)* checks if the ball (surrounded by *Bounding-Box)* hits on some walls. If a collision is detected by means of a switch, is seen with which direction the ball hits and based on it, it checks if the ball is approaching or moving away and resolves its address output. Likewise, updates the parameter friction to simulate the wear of the collision and make the ball to slow.

Collisions with moving objects. Balls

These collisions are equally easy to detect, as in the case of the walls, all the information is stored and updated, but they are difficult to simulate, because they must synchronize two or more state machines in the number of balls involved in the collision. The way to deal with these collisions is very similar to the case of the walls. In the state of treatment of collisions of the state machine (S2) we establish the conditional branches to verify all possible cases. If it detects that the position of the moving ball collides with another ball (which can stand or move, too), the program does the following:

1. Calculates the new direction of the ball.

2. Activates the relevant *flag* to *raise* adequate state machine.

3. Calculates the address you should start the ball impacted.

4. Go to the transition to S0.

All these points are made within a switch that is calculated based on the direction of the ball incident.

Should be noted that VHDL does not allow writing a signal from different processes, as the FPGA, when executes these processes concurrently, will find a conflict of multiple signals at the entrance to a record, etc. This force us to create different signals within each process, which has the function of modifying the signal that cannot be written from there. You can, however, read a signal from several sites, which translates into the same output device, connected to different sites. Therefore, to modify a signal from *outside,* create as many processes as auxiliary signals exist having to modify that signal. The inherent problems involved this is in excess of logic that may make some signals are lost, resulting in erroneous behavior.

Thus, the state machines of the balls that have been impacted have two possible answers:

1) If they were in the inactive state (S_ini) then they must read the *flag* on, which indicates the type of collisions they have suffered and the auxiliary signal to be loaded into its vector direction, load the new address and moving the state forward.

2) If they were in motion, they will detect the collision in the state *S1,* modify the direction and speed (significant increase of the parameter for this purpose), check that has not reached the threshold speed limit (in which case they should go to inactive state and stop) and continue execution.

The ball incidents also modify the parameter of friction to slow down and continuing the simulation.

Sound Module

The game makes a sound each time one of the balls collide with another ball or any of the bands of the table, and when the shot is made with the cue.

To make sounds with the FPGA, we use the audio CODEC Serial AK4520A wich is included in the FPGA. The Codec has the following inputs:

- *MCLK* or main clock.
- *LRCK* or channel selector.
- *SCLK* or clock of serial data transmission.
- *STDI* or serial input data.

Each of these signals is handled by its own counter to achieve the often necessary for the issuance of a note LA. Every time there is a connection, make a noise with a duration of 10 ^{6} cycles of the FPGA clock.

Conclusions

It has been developed a system of detection and treatment of dynamic collisions, capable of resolving collisions of mobile and static surfaces, which are managed through independent state machines, through a mechanism of warning *flags,* and applied to a Spanish pool game. It was also seen that the implementation of this mechanism, beyond the cost of maintaining the state machines, does not require much logic area of the FPGA, which can be adapted to more complex circuits and even to multiple FPGA systems.

Bibliography

[1] G. Baciu and S. K. Wong. Image-based techniques in a hybrid collision detector. In IEEE Trans. On Visualization and Computer Graphics, 2002.

[2] A. Gress and G- Zachmann. Object-space interference detection on programmable graphics hardware. In In SIAM Conf. On Geometric Design and Computing. 2003.

[3] D. Knott and D.K.Pai. Cinder: Collision and interference detection in real-time using graphics hardware. In Proc. Of Graphics Interface, 2003.

[4] Modelsim. http://www.model.com.

[5] Xilinx. http://www.xilinx.com.

[6] NAN Xi, GONG Longqing,TIAN Wei ,LI Xiao. Design and Implementation of Reconfigurable System Based on FPGA. Modern Electronics Technique, 2009

Authors' Information

Alejandro Figueroa Meana *- Facultad de Informática Universidad Complutense de Madrid, e-mail: afmeana@gmail.com*

Gustavo Méndez Muñoz *- Facultad de Informática Universidad Complutense de Madrid. e-mail:* gustavillo85@gmail.com

Francisco J. Cisneros de los Rios *– Natural Computing Group. Universidad Politécnica de Madrid, Boadilla del Monte, 28660 Madrid, Spain: e-mail:* kikocisneros@gmail.com

Adriana Toni – *Facultad de Informática Universidad Politécnica de Madrid. e-mail:* atoni@fi.upm.es