Design of an Algebraic Neural Operating System in Programmable Logical Controller for Simulation and Execution of Sequential Operations

Daniel M. Dubois * and Antonio Mascia **

* HEC Management School, N1, University of Liege,
rue Louvrex 14, B-4000 Liège, Belgium
Daniel.Dubois@ulg.ac.be - http://www.sia.hec.ulg.ac.be
and
CHAOS ASBL, Institute of Mathematics, B37,
Grande Traverse 12, B-4000 Liège, Belgium

** EURO VIEW SERVICES SA,
Chaussée de Lodelinsart 273, B-6060 Gilly, Belgium
management@euro-view.com - http://www.euro-view.com
and
ABEX EXPERT
http://www.mas1.be

Abstract
This paper deals with a general method for the analysis and the logical generation of discrete systems in Programmable Logical Controller (PLC). The Boolean operators are implemented with a generic and unique algebraic model as event-dependent discrete equations, which can be executed in a sequential order. With this method, a generator of sequential logical tables can be designed, simulated and executed for implementing discrete dynamical systems. Two applications are studied. The first application deals with the industrial automation of a water supply for a factory. From the logical table of the events, an algebraic model is designed with a set of discrete equations. From these digital equations, a neural network of the water supply is built. The second application deals with an industrial traveling wagon.
Keywords: control systems, computer systems, neural networks, logical controller, industrial automation

1. Introduction

This paper deals with industrial automation in relation with CAST (Pichler and Schwärtzel, 1992), and is the continuation of our work on the prototype GENSYSPRO (Dubois and Mascia, 1995a), (Dubois and Mascia, 1995b), and (Mascia and Dubois, 1995).

The purpose of this research is to design an algebraic neural operating system in PLC, Programmable Logical Controller, which automatically checks the logic of the

implemented discrete dynamical systems, for simulation and execution of sequential operations. What is the breakthrough with the prototype of the software GENSYSPRO, is the fact that it is at rest when no event happens, contrary to all the other industrial computing systems, which work all the time, based on an internal clock. GENSYSPRO is this an event-based software with a general method for the analysis and the logical generation of discrete systems in Programmable Logical Controller. The mathematical model is really a description of the dynamics of the process. Indeed, the model automatically permits the validation and the simulation of the process design without programming. The performances of the GENSYSPRO computing system for industrial automation are due to the fact that there is only an execution if there is a change detected by the XOR, either in the functional order (human decisions), either event based order. The methodology presented in this paper deals with an algebraic model of Boolean tables and the design of neural networks. The method to build the discrete equations of the Boolean tables depending on event steps permits to create automatically a neural system with McCulloch and Pitts formal neurons. Indeed, it is showed that non-linear digital equations can be easily built from Boolean Tables. These equations are Heaviside Fixed Functions that can be used to generate directly neural networks with McCulloch and Pitts formal neurons.

Several problems exist in the industrial automation systems, with the Programmable Logical Controller (PLC).
In the world, a completely safe 100% program of automation systems does not exist, and one can only program all combinatory possibilities.

For enhancing the procedure, the objectives consist to:

1. Reduce the global cost of automation.
2. Reduce the quantity of steps, time and tools.
3. Create completely safe programs and systems.
4. Create one unique graphic tool for the description, simulation, and execution and control the industrial process automation.
5. Increase the capabilities of the systems: execution speed.
6. Define a unique methodology (universality of program and training).

The automation problem is developed in only one global concept based on a generic model and a universal method.

The global concept permits the following functions:

A - Structured organization of the project (Object Oriented).
B - Graphical representation of the process.
C - Sequential description of the process.
D - Instantaneous simulation (by the operating system).
E – From the universal method to a unique training.

The universal method deals with the fact that the automation engineer:

1 - needs to describe only expected actions of the industrial automation system.
2 - describes sequentially all physical events that happened in the temporal process evolution.

Let us explain this, with a practical project described hereafter.

The global concept needs to create a new way of building the graphical software, the operating system and the conceptual construction of the industrial automation systems.

A graphical tool needs a complete object oriented database with an automatic variables generator.

The design tool, simulator tool and execution tool are regrouped in only one software tool: "all in one".

The global concept is based on the event management:

A first comparison by the logical exclusive OR (XOR) is realised on events on inputs.

A second comparison by XOR is realised on the functional action (example, on/off, auto/hand, local/remote, etc.). The functional commands events in all the systems are given by an initial sequence step, and many functional choices are activated or not activated. With XOR between a current functional choice and its change, the system will be able to activate or not activate actions and optimise the time of exploitation. The system works with active functions on event.

The operating system consists in the execution of the generic model that translates, in one CPU cycle, the Boolean matrix, representing the declarative sequences with inputs and outputs, to virtual algebraic equations for the output writing.

The methodology presented in this paper deals with an algebraic model of Boolean tables and the design of neural networks.

The method to build the discrete equations of the Boolean tables depending on event steps permits to create automatically a neural system with McCulloch and Pitts [1943] formal neurons. Indeed, one of us [Dubois, 1999] showed that non-linear digital equations are easily built from Boolean Tables.

These equations are Heaviside Fixed Functions that can be used to generate directly neural networks with McCulloch and Pitts formal neurons.

2. Theory of Algebraic Neural Networks of Dubois-Resconi

The Threshold Logic was initiated by the pioneer work of McCulloch and Pitts in 1943 [21], for modelling formal neurons at a logic level.

An extension of this Threshold Logic with non-linear argument in the Heaviside function of the formal neuron was published by D. M. Dubois and G. Resconi, in the Academy of Sciences of Belgium in 1993 [13]. Any truth tables can be modelled by non-linear neurons represented by Fixed Heaviside Functions.

The next section will recall the method to design neural networks with algebraic models.

2.1. Design of Algebraic Neural Systems of Boolean Tables

The content of this section is reprinted from the paper of D. M. Dubois (1999) [5].

Let us consider the general Boolean Table 1, with two inputs x_1 , x_2 and one output y. The values of the output y are given by the set $y = (y_1, y_2, y_3, y_4)$.

Table 1. General Table

x_1	x_2	y
0	0	y_1
0	1	y_2
1	0	y_3
1	1	y_4

Table 2a. XOR Table

x_1	x_2	y
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

Table 2b. AND Table

x_1	x_2	y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Table 2c. OR Table

x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

The following general algebraic equation

$$y = (1 - x_1).(1 - x_2).y_1 + (1 - x_1).x_2.y_2 + x_1.(1 - x_2).y_3 + x_1.x_2.y_4 \quad (1)$$

is a non-linear logic equation for the 16 Boolean Tables. The number of terms is equal to the number of lines in the Boolean table (2^n where n is the number of inputs). Each term is the product of the output value y_i, i=1,...,n, by all the inputs variables x_j or their complement $(1 - x_j)$ depending on its value 1 or 0 in the table at the line n. Here are examples for XOR, AND and OR, given in Figures 2abc:

$$y = (1 - x_1).x_2 + (1 - x_2).x_1 = x_1 + x_2 - 2.x_1.x_2 \quad (1a)$$

$$y = x_1.x_2 \quad (1b)$$

$$1 - y = (1 - x_1).(1 - x_2) \text{ or } y = x_1 + x_2 - x_1.x_2 \quad (1c)$$

The method for generating algebraic equations is general for any Boolean Table with any number of inputs and outputs.

McCulloch and Pitts formal neurons are defined as follows

$$y = \Gamma(w_1.x_1 + w_2.x_2 + w_3.x_3 + ... - \theta) \quad (2)$$

where w_i are the synaptic weights, θ the threshold and Γ is the Heaviside function defined by $\Gamma(x) = 0$ if $x \leq 0$ and $\Gamma(x) = 1$ if $x > 0$.

McCulloch and Pitts formal neurons can be built from the algebraic equations 1abc. Indeed, the terms given by products of the inputs or their complementary inputs can be represented by AND hidden neurons and the output by a OR neuron (the sum of all the AND hidden neuron). The AND neuron corresponding to eq. 1b is given by

$$y = \Gamma(x_1 + x_2 - 1) \quad (3a)$$

In a general way, an AND neuron of any equation given by a product $y = z_1.z_2....z_n$, is

$$y = \Gamma(z_1 + z_2 + z_3 + ... + z_n - (n-1)) \quad (3b)$$

where all the weights are equal to 1 and the threshold is equal to the number of inputs n minus 1. So each product of inputs in the digital equations can be represented by such AND neurons. These AND neurons will be hidden neurons, the outputs of which being the inputs of the output neuron y. The output neuron will be an OR neuron.
For generating the OR formal neuron from the algebraic eq. 1c, the following theorem is used [Dubois, 5]: For integer values of weights and threshold, the negation of the Heaviside function with the negation of its argument is equal to the Heaviside function of the argument

$$\Gamma(x) = 1 - \Gamma(1 - x) \quad (4)$$

for any integer x

From eq. 1c and eq. 3b, we can write

$$1 - y = (1 - x_1).(1 - x_2) = \Gamma(1 - (x_1 + x_2)) \quad (5a)$$

where the complement output 1–y is the AND of the complement inputs which is an AND formal neuron as shown previously. The eq. 5a can be written as

$$y = 1 - \Gamma(1 - (x_1 + x_2)) \quad (5b)$$

and, from eq. 4, we obtain the formal OR neuron

$$y = 1 - \Gamma(1 - (x_1 + x_2)) = \Gamma(x_1 + x_2) \quad (5c)$$

because the weights and the threshold are integers. So the OR is given by a Heaviside function with a linear sum of its inputs with weights equal to 1 and a null threshold.

In a general way, the OR neuron y for m inputs $y_1, y_2, ..., y_m$, is

$$y = \Gamma(y_1 + y_2 + y_3 + ... + y_m) \quad (5d)$$

Let us apply these relations to algebraic eq. 1 with two inputs and one output. This eq. 1 is a Heaviside Fixed Function and thus, we can define four AND hidden neurons y_1, y_2, y_3, y_4, corresponding to the 4 terms with products in eq. 1 and then one single output OR neuron:

$$y_1 = (1-x_1).(1-x_2).y_1 = \Gamma(-x_1 - x_2 + y_1) \quad (6a)$$

$$y_2 = (1-x_1).x_2.y_2 = \Gamma(-x_1 + x_2 + y_2 - 1) \quad (6b)$$

$$y_3 = x_1.(1-x_2).y_3 = \Gamma(x_1 - x_2 + y_3 - 1) \quad (6c)$$

$$y_4 = x_1.x_2.y_4 = \Gamma(x_1 + x_2 + y_4 - 2) \quad (6d)$$

$$y = y_1 + y_2 + y_3 + y_4 = \Gamma(y_1 + y_2 + y_3 + y_4) \quad (6e)$$

We remark that when $y_i = 0$, the argument of the Heaviside function is always null or negative, so the corresponding hidden neuron can be cancelled. The weights are –1 or +1 when the corresponding value of the input is 0 or 1, and the threshold is equal to the

sum of all the input values minus the output value for each line of the Boolean table. The outputs of the hidden neurons are mutually exclusive, $y_i \cdot y_j = 0$ for $i \neq j$. The weights of the output neuron are 1 and the threshold is 0, so the output neuron is the sum of the outputs of the hidden neurons.

For example, XOR Boolean table 2a can be represented by two hidden neurons y_2 and y_3 for which $y_2 = y_3 = 1$, so XOR neural network is given by two input neurons, x_1 and x_2, two hidden neurons y_2 and y_3, given by the two following Heaviside threshold functions

$$y_2 = \Gamma(- x_1 + x_2) \tag{7a}$$

$$y_3 = \Gamma(+ x_1 - x_2) \tag{7b}$$

and one single output neuron y, given by the following Heaviside threshold function

$$y = \Gamma(+ y_2 + y_3) \tag{7c}$$

2.2. Design of the Neural Network of the XOR Boolean Table

The figure 1 gives the XOR neural network, based on the eqs. 7abc.

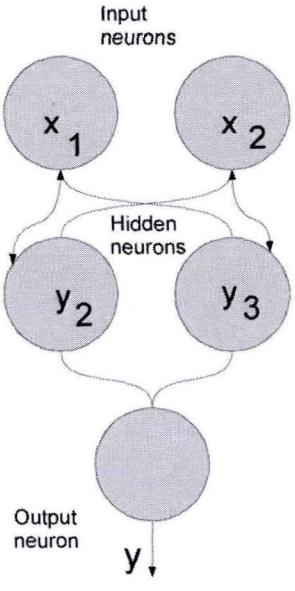

Figure 1. The XOR neural network is given by eqs. 7a-b-c, with weights, +1 and −1, and a null threshold.

Let us apply this methodology to two industrial applications.

3. The Algebraic Model of a Water Supply

This section describes the project for the industrial automation of the process of a water supply for a factory.

The analysis and graphical description of the process of the water supply for a factory is given in Figure 2, which gives a start condition without water in the tank.

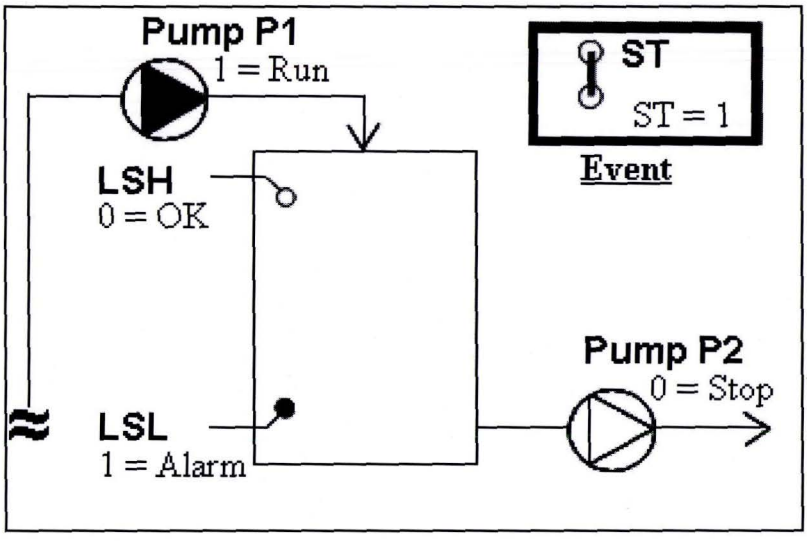

Figure 2: This figure corresponds to a start condition, with no water in the tank, where the Event is the Start/Stop at ST = 1, and the Pump P1 at 1 = Run.

The abbreviations are given as follows in the analysis and graphics of the water supply.:

ST: Start/Stop switch water supply, 0 is Stop and 1 is Start
P1: Pump 1 (input tank), 0 is Stop pump and 1 is Run pump
P2: Pump 2 (output tank), 0 is Stop pump and 1 is Run pump
LSL: Level Switch Low, 0 is OK (no alarm - white) and 1 is Alarm low level (black)
LSH: Level Switch High, 0 is OK (no alarm - white) and 1 is Alarm high level (black)

Let us describe the different steps of system evolution on the graphics:

- The description is done with graphic tools
- The program translates this description into Boolean data matrix

The following 4 figures 3-ABCD, give the graphical design, with their variables and steps, where the "Event" is shown.

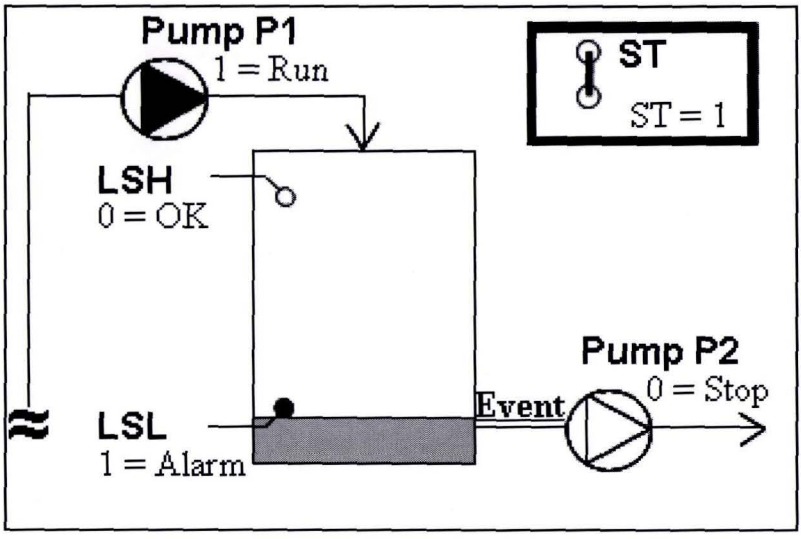

Figure 3-A: Event 1 representing the Low Level, with LSL at 1 = Alarm, and the Pump P1 at 1 = Run.

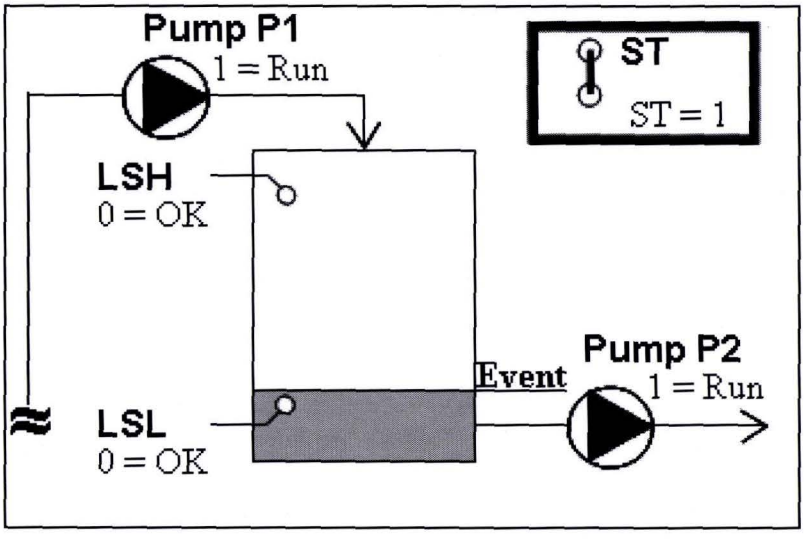

Figure 3-B: Event 2 representing the Normal Level, with LSL at 0 = OK, and the Pump P2 at 1 = Run.

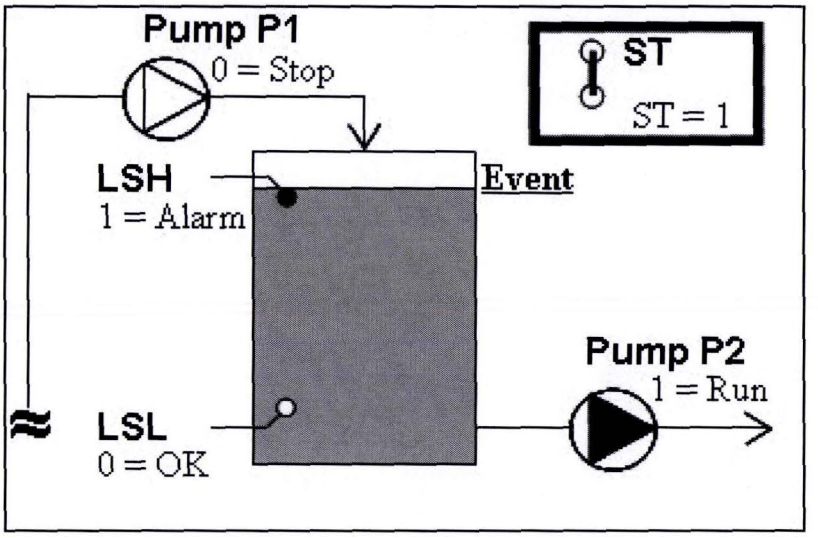

Figure 3-C: Event 3 representing the High Level, with LSH at 1 = Alarm, and the Pump P1 at 0 = Stop.

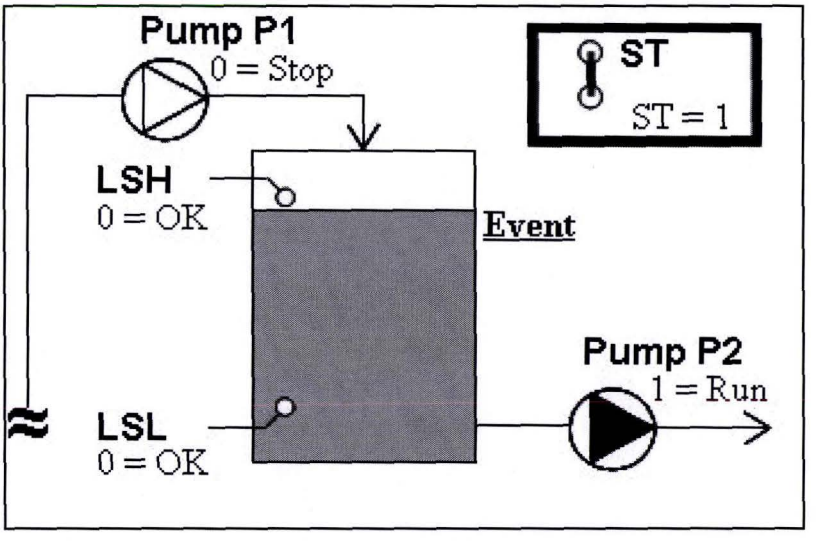

Figure 3-D: Event 4 representing the Normal Level, with LSH at 0 = OK.

Let us now give the Boolean presentation of this process. The validation of the evolution is given by the simulation mode. The program creates the whole table before the model generation. The future occurrences, at event step k+1, are taken into account for anticipation, and the past occurrences, at event steps k–1, k–2, ... are taken into account to suppress incoherencies.
The following table 3 gives the logical table of the successive event steps k–1, k, k+1.

Table 3: Logical table of the events cycle of the water supply

		Inputs					Outputs	
Step k	**Levels**	**ST**	**LSL(k)**	**LSH(k)**	**LSL(k–1)**	**LSH(k–1)**	**P1**	**P2**
1	Low level	1	1	0	0	0	1	0
2	Normal level	1	0	0	1	0	1	1
3	High level	1	0	1	0	0	0	1
4	Normal level	1	0	0	0	1	0	1

Important Remark: The event step index "k–1, k, k+1" represents the numbering of steps of the events, and does not represent the time interval $\Delta t(k)$ between two successive events, which is not necessarily a constant. The index k becomes k+1, at each new event. For a cycling process, the next k is k = 1, when k = k_{max} is the maximum number of steps of the process. So for 4 cycling steps, the values of k are given by k = 1, 2, 3, 4, 1, 2, 3, 4, … , so the next, k, is given by, 1 + (k modulo 4).

The model consists in a set of algebraic equations, as explained in section 2.

Here are the algebraic equations for the water supply for each output P1 and P2:

$$P1=ST.LSL.(1-LSH).(1-LSL_{k-1}).(1-LSH_{k-1})$$
$$+ST.(1-LSL).(1-LSH).LSL_{k-1}.(1-LSH_{k-1}) \quad (8a)$$

$$P2=ST.(1-LSL).(1-LSH).LSL_{k-1}.(1-LSH_{k-1})$$
$$+ST.(1-LSL).LSH.(1-LSL_{k-1}).(1-LSH_{k-1})$$
$$+ST.(1-LSL).(1-LSH).(1-LSL_{k-1}).LSH_{k-1} \quad (8b)$$

GENSYSPRO shows three functions in one step: the application design (graphic), with the simulator system, and with the supervision and control system. The result goes to the PLC and executes the algebraic equations of this model.

Next section will give the design of the neural network corresponding to this model of the water supply.

3.1. Design of the Neural Network of the Water Supply

From the algebraic model of the water supply, given by the two equations 3-ab, let us build the neural network, with the method described in section 2.

The input neurons are given by the following variables:

ST, LSL, LSH, LSL_{k-1}, and LSH_{k-1}

The threshold functions of the 2 hidden neurons for P1 are given by

$$H11 = \Gamma(ST + LSL - LSH - LSL_{k-1} - LSH_{k-1} - 1) \quad (9a)$$

$$H12 = \Gamma(ST - LSL - LSH + LSL_{k-1} - LSH_{k-1} - 1) \quad (9b)$$

The threshold functions of the 3 hidden neurons for P2 are as follows

$$H21 = \Gamma(ST - LSL - LSH + LSL_{k-1} - LSH_{k-1} - 1) \quad (10a)$$

$$H22 = \Gamma(ST - LSL + LSH - LSL_{k-1} - LSH_{k-1} - 1) \quad (10b)$$

$$H23 = \Gamma(ST - LSL - LSH - LSL_{k-1} + LSH_{k-1} - 1) \quad (10c)$$

The threshold functions of the 2 output neurons are finally represented by

$$P1 = \Gamma(H11 + H12) \quad (11a)$$

$$P2 = \Gamma(H21 + H22 + H23) \quad (11b)$$

The figure 4 gives the neural network corresponding to the water supply.

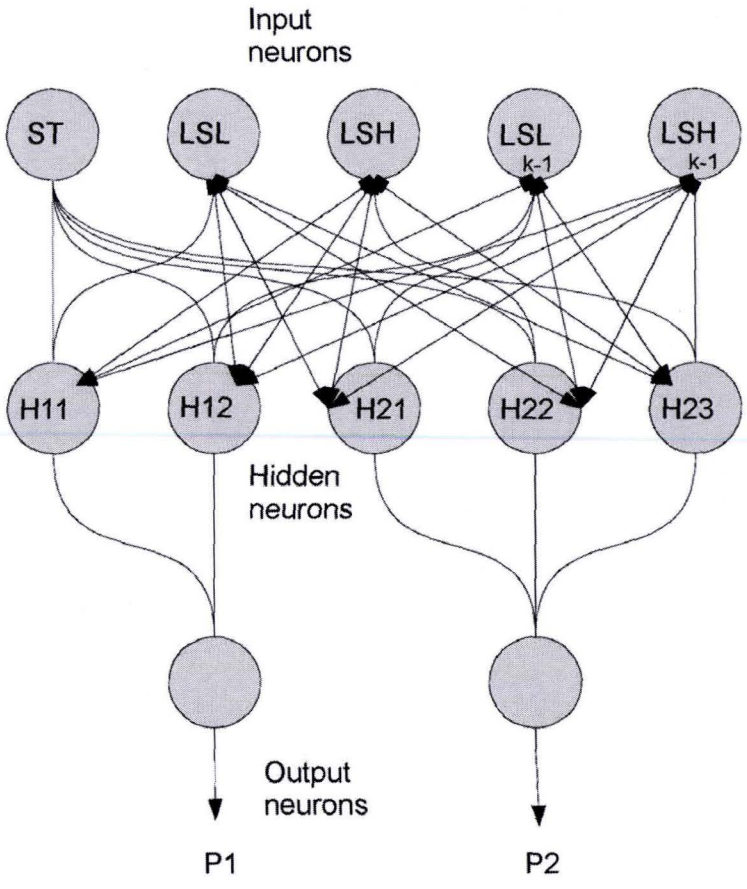

Figure 4. Neural Network corresponding to the algebraic model of the water supply application.

4. Model of a Travelling Wagon

Here is another industrial example with a specific logical problem given by an industrial wagon travelling between two positions: at the left, SPS is the Start Position Sensor, as shown in figure 5. The sequential description is given in this figure 5.

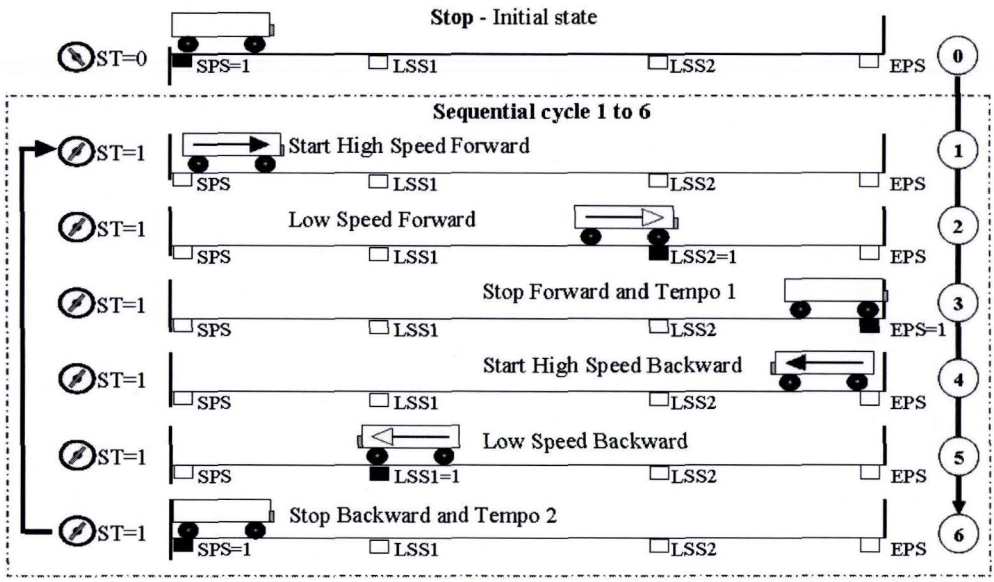

Figure 5: Sequential cycle description of travelling wagon

The meaning of the abbreviations used with this example is given in the table 4.

Table 4. Abbreviations used for the travelling wagon

Translation:	**Speed:**
1- Forward 2- Backward	1- High 2- Low
Inputs:	**Outputs:**
ST: Start/Stop button SPS: Start Position Sensor EPS: End Position Sensor LSS1: Low Speed Sensor 1 LSS2: Low Speed Sensor 2 T1: end of Tempo 1 T2: end of Tempo 2	LSF: Low Speed Forward HSF: High Speed Forward LSB: Low Speed Backward HSB: High Speed Backward T1: Start Tempo 1 T2: Start Tempo 2

The successive steps are given in table 5.

Table 5: Successive steps of the travelling wagon

		INPUTS							OUTPUTS			
k		**SPS**	**LSS1**	**LSS2**	**EPS**	**T1**	**T2**	**ST**	**LSF**	**HSF**	**LSB**	**HSB**
1	**SPS=1**	1	0	0	0	0	0	1	0	0	0	0
2	**T1=1**	1	0	0	0	1	0	1	0	1	0	0
3	**SPS=0**	0	0	0	0	0	0	1	0	1	0	0
4	**LSS1=1**	0	1	0	0	0	0	1	0	1	0	0
5	**LSS1=0**	0	0	0	0	0	0	1	0	1	0	0
6	**LSS2=1**	0	0	1	0	0	0	1	1	0	0	0
7	**LSS2=0**	0	0	0	0	0	0	1	1	0	0	0
8	**EPS=1**	0	0	0	1	0	0	1	0	0	0	0
9	**T2=1**	0	0	0	1	0	1	1	0	0	0	1
10	**EPS=0**	0	0	0	0	0	0	1	0	0	0	1
11	**LSS2=1**	0	0	1	0	0	0	1	0	0	0	1
12	**LSS2=0**	0	0	0	0	0	0	1	0	0	0	1
13	**LSS1=1**	0	1	0	0	0	0	1	0	0	1	0
14	**LSS1=0**	0	0	0	0	0	0	1	0	0	1	0

There are incoherencies in this table 5, because some outputs are different for the same inputs.
As for the water supply example, the step k−1 is taken into account to avoid the logical incoherence.
But in this case, it appears that outputs are still different for the same inputs, in two successive steps, k−1 and k.
There are two possibilities to correct the problem, either to take the time t−2 into account, or to add an input variable FW (Forward).
This last solution is chosen, because it only adds one more variable FW in the algebraic model. FW is defined using the feedbacks of the outputs and the start button, as shown in table 6.

Table 6. Behaviour of FW.

	INPUTS				OUTPUT
ST	**LSF**	**HSF**	**LSB**	**HSB**	**FW**
1	1	0	0	0	1
1	0	1	0	0	1

The following table 7 is the table 5 built with the variable FW as INPUT, and also in adding T1 and T2 as OUPUTS.

Table 7: Logical table of the travelling wagon

		INPUTS								OUTPUTS					
k		**SPS**	**LSS1**	**LSS2**	**EPS**	**T1**	**T2**	**ST**	**FW**	**LSF**	**HSF**	**LSB**	**HSB**	**T1**	**T2**
1	**SPS=1**	**1**	0	0	0	0	0	**1**	0	0	0	0	0	**1**	0
2	**T1=1**	**1**	0	0	0	**1**	0	**1**	0	0	**1**	0	0	0	0
3	**SPS=0**	0	0	0	0	0	0	**1**	**1**	0	**1**	0	0	0	0
4	**LSS1=1**	0	**1**	0	0	0	0	**1**	**1**	0	**1**	0	0	0	0
5	**LSS1=0**	0	0	0	0	0	0	**1**	**1**	0	**1**	0	0	0	0
6	**LSS2=1**	0	0	1	0	0	0	**1**	**1**	**1**	0	0	0	0	0
7	**LSS2=0**	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0
8	**EPS=1**	0	0	0	1	0	0	**1**	**1**	0	0	0	0	0	**1**
9	**T2=1**	0	0	0	1	0	1	**1**	**1**	0	0	0	**1**	0	0
10	**EPS=0**	0	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0
11	**LSS2=1**	0	0	**1**	0	0	0	**1**	0	0	0	0	**1**	0	0
12	**LSS2=0**	0	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0
13	**LSS1=1**	0	**1**	0	0	0	0	**1**	0	0	0	**1**	0	0	0
14	**LSS1=0**	0	0	0	0	0	0	**1**	0	0	0	**1**	0	0	0

On k = 9, Tempo 2 is set to the wanted time duration, where T2 = 1 at the elapsed time.

On k = 2, Tempo 1 is set to the wanted time duration, where T1 = 1 if the duration is finished.

The algebraic model is given by the following 6 digital equations 12-abcdef.

LSF=1 for k=6 or k=7, LSF=0 for all the other values of k:

LSF=(1−SPS).(1−LSS1).LSS2.(1−EPS).(1−T1).(1−T2).ST.FW.(1−SPS[k−1]).
(1−LSS1[k−1]).(1−LSS2[k−1]).(1−EPS[k−1]).(1−T1[k−1]).(1−T2[k−1]).ST[k−1].
FW[k−1] +
(1−SPS).(1−LSS1).(1−LSS2).(1−EPS).(1−T1).(1−T2).ST.FW.(1−SPS[k−1]).
(1−LSS1[k−1]).LSS2[k−1].(1−EPS[k−1]).(1−T1[k−1]).(1−T2[k−1]).ST[k−1].FW[k−1]

(12a)

HSF=1 for k=2 or k=3 or k=4 or k=5, HSF=0 for all the other values of k:

HSF=SPS.(1–LSS1).(1–LSS2).(1–EPS).T1.(1–T2).ST.(1–FW).SPS[k–1].
(1–LSS1[k–1]).(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
(1– FW[k–1]) +
(1–SPS).(1–LSS1).(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.FW.SPS[k–1].
(1–LSS1[k–1].(1–LSS2[k–1]).(1–EPS[k–1]).T1[k–1].(1–T2[k–1]).ST[k–1].
(1–FW[k–1]) +
(1–SPS).LSS1.(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.FW.(1–SPS[k–1]).
(1–LSS1[k–1]).(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
FW[k–1] +
(1–SPS).(1–LSS1).(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.FW.(1–SPS[k–1]).
LSS1[k–1].(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].FW[k–1]

(12b)

LSB=1 for k=13 or k=14, LSB=0 for all the other values of k:

LSB=(1–SPS).LSS1.(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.(1–FW).(1–SPS[k–1]).
(1–LSS1[k–1].(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
(1–FW[k–1]) +
(1–SPS).(1–LSS1).(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.(1–FW).(1–SPS[k–1]).
LSS1[k–1].(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
(1–FW[k–1]) (12c)

HSB=1 for k=9 or k=10 or k=11 or k=12, HSB=0 for all the other values of k:

HSB=(1–SPS).(1–LSS1).(1–LSS2).EPS.(1–T1).T2.ST.FW.(1–SPS[k–1]).
(1–LSS1[k–1]).(1–LSS2[k–1]).EPS[k–1].(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
FW[k–1] +
(1–SPS).(1–LSS1).(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.(1–FW).(1–SPS[k–1]).
(1–LSS1[k–1]).(1–LSS2[k–1]).EPS[k–1].(1–T1[k–1]).T2[k–1].ST[k–1].FW[k–1] +
(1–SPS).(1–LSS1).LSS2.(1–EPS).(1–T1).(1–T2).ST.(1–FW).(1–SPS[k–1]).
(1–LSS1[k–1]).(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
(1–FW[k–1]) + (1–SPS).(1–LSS1).(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.(1–FW).
(1–SPS[k–1]).(1–LSS1[k–1]).LSS2[k–1].(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).
ST[k–1].(1–FW[k–1]) (12d)

T1=1 for k=1, T1=0 for all the other values of k:

T1=SPS.(1–LSS1).(1–LSS2).(1–EPS).(1–T1).(1–T2).ST.(1–FW).(1–SPS[k–1]).
(1–LSS1[k–1]).(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
(1–FW[k–1]) (12e)

T2=1 for k=8, T2=0 for all the other values of k:

T2=(1–SPS).(1–LSS1).(1–LSS2).EPS.(1–T1).(1–T2).ST.FW.(1–SPS[k–1]).
(1–LSS1[k–1]).(1–LSS2[k–1]).(1–EPS[k–1]).(1–T1[k–1]).(1–T2[k–1]).ST[k–1].
FW[k–1] (12f)

These equations are Heaviside Fixed Functions and can be used to design a neural network, as already shown in the section of the water supply.

5. CONCLUSION

This work permitted to outline some properties in industrial automation for developing a new industrial operating system (IOS), which gives rise to a semantic information about the process. The main point is that such a CAST, Computer Aided Systems Theory for the Design of Intelligent Machines, presented in this paper, would open new avenues where programmation and artificial languages would disappear in profit of the Human Natural Language.

All the actual computing systems work on a permanent cyclic recursive basis, without necessarily execute a function. Nevertheless, all the computing systems execute functional orders given by man and execute an output value only if the equation is true based on an event (evolution) of an input or on a parameter that has changed.

GENSYSPRO does execute nothing if there is no change of functional order (controlled by an XOR), and if there is no event or change of parameters (XOR on the inputs). What is the breakthrough with GENSYSPRO, is the fact that it is at rest when no event happens, contrary to all the other industrial computing systems, which work all the time, based on an internal clock. GENSYSPRO is thus an event-based software with a general method for the analysis and the logical generation of discrete systems in Programmable Logical Controller. The performances of the GENSYSPRO computing system for industrial automation are due to the fact that there is only an execution if there is a change detected by the XOR, either in the functional order (human decisions), either event based order.

This approach deals with artificial intelligence, neural networks and, recently, with multi-agent systems [Wooldridge, 2009].

A new type of operating systems can be designed, which automatically checks the logic of the implemented discrete dynamical systems, for simulation and execution of sequential operations. The mathematical model is really a description of the dynamics of the process. Indeed, the model automatically permits the validation and the simulation of the process design without programming. The Boolean operators are implemented with a generic and unique algebraic model as event-dependent discrete equations, which can be executed in a sequential order. So, a generator of sequential logical tables can be designed, simulated and executed for implementing discrete dynamical systems. Research is in progress to develop such an operating system based on neural systems generated by algebraic models based on logical tables described in natural language. This is a very important field in risk engineering management.

References

1. Dubois D. M., Resconi G. (1992), HYPERINCURSIVITY: a new mathematical theory, Presses Universitaires de Liège.
2. Dubois D. M., Resconi G. (1993), Mathematical Foundation of a Non-linear Threshold Logic: a new Paradigm for the Technology of Neural Machines, ACADEMIE ROYALE DE BELGIQUE, Bulletin de la Classe des Sciences, 6ème série, Tome IV, 1-6, pp. 91-122.
3. Dubois D. M. (1995), "Modelling of Fractal Neural Networls". In Proceedings of the 14th International Congress on Cybernetics, Namur (Belgium), 21st-25th August 1995, publ. by International Association for Cybernetics, pp. 405-410
4. Dubois D. M. (1998), "Boolean Soft Computing by Non-linear Neural Networks with Hyperincursive Stack Memory". In *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*, Edited by O. Kaynak, L. A. Zadeh, B. Türksen, I. J. Rudas, NATO ASI Series, Series F: Computer and Systems Sciences, volume. 162, Springer-Verlag.
5. Dubois Daniel M. (1999) Hyperincursive McCulloch and Pitts Neurons for Designing a Computing Flip-Flop Memory. Computing Anticipatory Systems: CASYS'98 - Second International Conference. Edited by Daniel M. Dubois, Published by The American Institute of Physics, AIP Conference Proceedings 465, pp. 3-21.
6. Dubois D. M., A. Mascia (1995), Méthode Générale pour l'Analyse et la Programmation des Systemes Discrets, Proceedings of the 14th International Congress on Cybernetics, edited by the International Association for Cybernetics, pp.571-576.
7. Dubois D. M. and A. Mascia (1995), Computer Aided Generator of Models for Designing Automation Devices, in Advances in Computer Cybernetics, volume III, edited by G. E. Lasker, published by The International Institute for Advanced Studies in Systems Research and Cybernetics, University of Windsor, Canada, pp. 23-27.
8. Mascia A., Dubois D. M. (1995), Générateur de Modèles Algébriques Appliqué à la Conception d'un Automatisme Industriel, Proceedings of the 14th International Congress on Cybernetics, edited by the International Association for Cybernetics, pp.577-582..
9. McCulloch W. S., Pitts W. (1943), A logical calculus of the ideas immanent in nervous activity, Bulletin of mathematical Biophysics, vol 5, pp. 115-133.
10. Pichler F., H. Schwärtzel (Eds.) (1992), CAST Methods in Modelling, Computer Aided Systems Theory for the Design of Intelligent Machines, Springer-Verlag, Berlin, Heidelberg.
11. Wooldridge Michael (2009). An Introduction to MultiAgent Systems, John Wiley & Sons, 2nd Edition