18

A Multiagent Architecture Based in aFoundation Fieldbus Network Function Blocks

Vinicius Ponte Machado
Federal University of Piaui
Natal – RN – Brazil
Dennis Brandão
University of São Paulo
São Carlos – SP – Brazil
Adrião Duarte Dória Neto
Federal University of Rio Grande do Norte
Natal – RN – Brazil
Jorge Dantas de Melo
Federal University of Rio Grande do Norte
Natal – RN – Brazil

1. Introduction

The industrial automation is directly related to the technological development of information. Better hardware solutions, as well as improvements in software development methodologies have made possible the rapid development of the productive process control. In this Chapter, it is proposed an architecture that permits to join two technologies in the same hardware (Industrial Network) and software context (Multiagent Systems – MAS). We show a multiagent architecture which uses an algorithm-based Artificial Neural Network (ANN) to learn about fault problem patterns, detect faults, and adapt algorithms that can be used in these fault situations. We also present a dynamic Function Block (FB) parameter exchange strategy which allows agent allocation in fieldbus. This proposed architecture reduces the supervisor intervention to select and implement an appropriate structure of function block algorithms. Furthermore, these algorithms, when implemented into device function blocks, provide a solution at fieldbus level, reducing data traffic between gateway and device, and speeding up the process of dealing with the problem. We also present some examples for our approach. The first one introduces FBSIMU which simulates Foundation Fieldbus function blocks architecture. This software has a controlled process and allocates the MAS to detect and correct faults. The second example shows a multiagent architecture that implements the neural network change in a laboratory test process which imitates fault scenarios.

2. Theoretical Foundation

2.1 Foundation Fieldbus Protocol

The term FOUNDATION Fieldbus (FF) indicates the protocol specified by the Fieldbus Foundation and standardized by IEC[1] standards number 61158 (IEC, 2000) and 61784 at profile CPF[2] - 1/1 (IEC, 2003). It is a digital, serial, bidirectional, and distributed protocol, which interconnects field devices such as sensors, actuators and controllers. Basically, this protocol can be classified as a LAN (Local Area Network) for instruments used in process and industrial automation, with the ability to distribute the control application through a network.

This protocol is based on the ISO/OSI (International Organization for Standardization/Open System Interconnection) seven layer reference model (ISO, 1994). Although being based on the ISO/OSI model, the FF does not use the network layer, the transport layer, the section layer, nor the presentation layer, because it is restricted to local applications. The entire network structure of the FF concentrates on the physical layer, the data link layer (DLL) and the application layer. Besides these three implemented layers, the protocol defines an additional layer named User Application Layer.

The FF Physical Layer, named H1, uses a shielded twisted pair cable as communication medium. The H1 specifies a 31.25 kBit/s bit rate with Manchester codification over a bus powered channel. The network topology configuration is flexible: it is typically configured with a trunk and several spurs, attending certain physical and electrical limitations regarding maximum spur lengths and number of transmitters.

The DLL carries the transmission control of all messages on the fieldbus and its protocol grants to the FF network temporal determinism for critic process control data. The communication is based on a master–slave model with a central communication scheduler (master), named Link Active Scheduler or LAS. This node performs the medium access control (MAC). Two types of DLL layer are standardized: Basic and Link Master. A Basic DLL transmitter does not have LAS capabilities, it operates passively as a communication slave. A Link Master DLL transmitter, on the other hand, can execute LAS functions and thus, if the active LAS node fails, become the LAS node. The FF Data Link Layer supports two transmission policies: one addressed to scheduled cyclic data, and another to sporadic (unscheduled) background data. These two communication policies share the physical bus, but they are sequentially segmented in cyclic time slots or periods. In the scheduled communication period, most process variables generated by periodic processes are transmitted cyclically according to a static global schedule table loaded on the LAS node. This cyclic transmission mode has higher priority over acyclic transmission modes. A periodic process can be defined as a process initiated at predetermined points in time, also called a time-triggered process.

The period for the network cyclic process is typically from tenths to hundreds of milliseconds, and it is mandatory to consider that the generated data must be delivered before the next data is available. This type of periodic data is usually related to measurement and control variables (Cavalieri et al., 1993).

[1] International Electrotechnical Commission

[2] Communication *Profile* Family

Sporadic or unscheduled communication is used to transmit non periodic, or aperiodic, data, generated by sporadic processes not directly related to the control loop cycles, but to user configuration actions and data supervision efforts. The unscheduled transmissions are dispatched under a token pass scheme. A token that circulates among all active nodes on the bus is used in FF protocol.

Once a transmitter receives the token, it is granted the right to send pending aperiodic messages with a minimum priority for a specific time period. Non periodic (or event-triggered) processes are initiated as soon as specific events are noted (Pop et al., 2002). The event-triggered processes are unpredictable and usually related to alarm notifications, configuration data and user commands as cited before. Although acyclic traffic is less frequent than the cyclic one, the acyclic data should also be delivered prior to a given deadline, according to the system requirements. For a description of the MAC operation on both cyclic and acyclic phases, refer to Hong & Ko (2001), Wang et al. (2002), Petalidis & Gill (1998).

The FF User Layer is directly related to the process automation tasks, and it is based on distributed control or monitoring strategies composed of Function Blocks (FB). Function Blocks are User Layer elements that encapsulate basic automation functions and consequently make the configuration of a distributed industrial application modular and simplified (Chen et al., 2002). Distributed among the transmitters, the FBs have their inputs and outputs linked to other blocks in order to perform distributed closed control loop schemes. When blocks from different transmitters are linked together, a remote link is configured and mapped to a cyclic message. Considering that all cyclic messages should be released in a predetermined instant defined on a schedule table, and that they carry data generated by the FBs, it is adequate to synchronize the execution of the FB set on the system with the referred cyclic transmissions schedule table. This solution leads to the concept of joint scheduling (Ferreiro et al., 1997).

The Foundation Fieldbus standardized a set of ten basic function blocks (Fieldbus Foundation, 1999a), a complementary set of eleven advanced control blocks (Fieldbus Foundation, 1999b), and a special flexible function block intended to be fully configurable by the user, i.e., internal ladder logic and parameter set (Fieldbus Foundation, 1999c). The standard and advanced block sets provide mathematical and engineering calculations necessary to configure typical industrial control loop strategies, while the flexible function block can be applied to custom or advanced controls or to complex interlocking logics based on ladder nets. It is important to mention, however, that the standard is open at this point, permitting the integration of ''user-defined'' custom function blocks in order to enhance the capabilities of FF control system, and make the integration of novel control techniques possible.

2.2 Multiagent systems

An agent is a computer system, a paradigm to the development of software applications that is situated in some environment and is capable of autonomous action in that environment in order to meet its design objectives (Russell & Norvig, 2003). In a few words, a multiagent system (MAS) is a problem of placing the agents together (organized as a society). The application components of a multiagent system are agents. Several different multiagent architectures can be found in the literature including applications in automation (Weyns et al., 2005; Weyns & Holvoet, 2007; Seilonen et al., 2002; Feng et al., 2007). In a MAS, control is decentralized, i.e., none of the system components has global control over the system or global knowledge about the distributed system.

The main reason for the use of agents in these environments is that these applications need distributed interpretation and distributed planning by means of intelligent sensors.

Furthermore, distributed multiagent systems are an appropriate concept for many fields of industrial automation like monitoring, fault diagnosis, simulation and control, as they give several advantages for these applications. They allow distributed data collection while maintaining a high level of scalability and flexibility, once they keep network load low through an adequate pre-processing. They also provide on-site reactivity and intelligence that is required in various remote control scenarios, since the network channel is not capable of transporting each and every control command. Finally they offer an abstraction level when accessing proprietary devices for monitoring and control, and they are often easier to integrate into existing applications than, for example, a service oriented architecture (Theiss et al., 2008).

Applications of agent technology in the research of process automation systems have not been as numerous as many other industrial application domains. Neither the way to apply agent technology in process automation nor the possible utility of it has been so evident in process automation than in other fields. However, some promising research has been reported and some experiences from other fields might also have applicability in process automation (Seilonen et al., 2002).

Autonomy, high encapsulation and reactivity of agents motivate their usage in large automation systems. The application area of multiagent systems includes power supply systems, manufacturing systems, building automation and mobile applications (Jennings & Bussmann, 2002). The agent functionality comprises monitoring and diagnosis (e.g., Taylor & Sayda, 2003; Albert et al., 2003; Pirttioja et al., 2005), control, scheduling, modelling and simulation of these applications. The agents primarily operate on management level (Schoop et al., 2002) and use web-based technologies like web services and OSGi (Fei-Yue et al., 2005). This allows them to use the PCs and servers as hosts, making the performance, memory usage and real-time issues negligible.

As a conclusion to the current research about agent applications in process automation and other control applications, one could state that agents have generally been applied either for higher-level, non real-time and event-based operations, or for integration purposes. The state-of-the-art research regarding MAS applications in process automation leaves some questions unanswered behind. Research has mainly focused on control functions and the functional role of MAS in process automation. Other functions, e.g., monitoring and information access, have received less attention (Seilonen, 2006). Furthermore, in many cases these models do not address the issue of deterministic response times. Unlike the aforementioned studies, we have shown MAS architecture which enables the implementation of control configuration at the fieldbus level. We believe this is the main contribution of our work. A similar study was proposed by Brennan et al. (2002). However, in our study we aggregate machine learning through an Artificial Neural Network (ANN) and FIPA (Foundations of Intelligent Physical Agents) compliant agents. Moreover, our implementation raises a basic agent feature: adaptation. The function block allocation will change to adapt to a type of problem, without user intervention.

3. Function Block Intelligent Algorithms

Smart configuration strategies are implemented by intelligent algorithms that are incorporated to the sensors using the standard function blocks. These blocks have basic functions that, when combined, are able to implement the artificial neural network for example. The organization of function blocks is essential to the success of this type of process.

The protocol found to better suit these demands was the Foundation Fieldbus protocol, because its system is gifted with the capacity of distributing the control of the process in the field, i.e. the sensors and actuators have embedded processors which can execute the algorithms in a distributed way.

Many projects have been developed using Foundation Fieldbus protocol and function blocks. In ours laboratories (LAMP - Petroleum Measurement and Evaluation Laboratory in Federal University of Rio Grande do Norte) some intelligent algorithms were implemented using mainly neural networks.

In Silva et al. (2006), we can see a solution to execute artificial neural network algorithms in the environment of networks to Foundation Fieldbus industrial automation, based on standardized function blocks. This strategy involves two function blocks: arithmetic and characterizer. They must be configured and linked in such a way that the set behaves as an artificial neuron (Haykin, 1999). In the arithmetic block, the Algorithm Type parameter must be chosen as Traditional Adder and the gains of the inputs must be filled according to the training performed, as well as the bias values.

By linking the output of an arithmetic function block to the input of a signal characterizer function block, configured as described above, we have an artificial neuron in the FF environment. And by linking of these neurons, neural networks are built.

With another neural network function block, configuration of the agents can compensate the error as it is seen in Cagni et al. (2005). In his work the implementation of the self-calibration, self-compensation and self-validation algorithms for Foundation Fieldbus sensors are presented using standard function blocks.

The deterioration of the sensors can make the sensor measurement precision decrease in time, until another calibration of the sensor is made. The lack of precision and the calibration process can be economically disadvantageous to the industries. Determining what is the best calibration period for a sensor is the main focus of interest of this research. With this purpose, some based-neural network algorithms were created to increase the precision and the reliability of data collected by the sensors and to optimize the calibration periods. These algorithms are the self-compensation, self-calibration and self-validation ones.

The addition of noise is another very common problem during the process of extracting information generated by a sensor installed in a field network. In Costa et al. (2003), the implementation of a system is proposed so that, beginning from software embedded in a DSP (Digital Signal Processor), interacts with fieldbus devices connected through a Foundation Fieldbus network. This approach, based on the technique of Independent Component Analysis (ICA), presents an efficient solution to the problem of extraction of the noise derived from the sensor. In other work, Fernandes et al. (2007) presents an approach to process fault detection and isolation (FDI) system applied to a level control system connected with an industrial network Foundation Fieldbus. The FDI system was developed using artificial neural networks (ANN). Basically, the FDI system was divided in two parts:

the first corresponds to neural identification of the plant model; and the second, to the detection and isolation of faults in process.

4. Foundation Fieldbus Simulated Environment

The basic concept of the FBSIMU (Foundation Fieldbus Simulated Environment) architecture is to map each Function Block, as well as the plant, in an independent LabVIEW[3] application, also named Virtual Instrument (VI). The configuration of the whole system is centralized in the FBSIMU.CONF module. This module's graphical user interface is inspired by commercial fieldbus configuration tools. As mentioned before, the FBSIMU is focused on the function block application layer and it is composed exclusively of software according to a modular and extensible architecture. The simulator was developed in LabVIEW using the G graphical programming language, ''native'' language in this environment. Each FBSIMU module or software unit simulates an element or a structure of a real FOUNDATION Fieldbus system (Brandão, 2005).

4.1 Function Block simulation

The Function Block modules are programmed into the FBSIMU according to the FF specifications directions and, consequently, the usage and configuration of a simulated control loop on the FBSIMU environment is identical to a real FF system. A ''LabVIEW Foundation Fieldbus Tool Kit'' library has been developed (Pinotti et al., 2005) to provide a range of typical Foundation Fieldbus control and acquisition functions, according to the standards. These functions encapsulate different FF calculations and data type manipulations necessary to build standard or custom Function Blocks. A Function Block seed module is also used to accelerate the process of developing and integrating new projects. The seed has the whole FB module structure (an empty structure) and directions to proceed with a FB project from the design to the final test procedures.

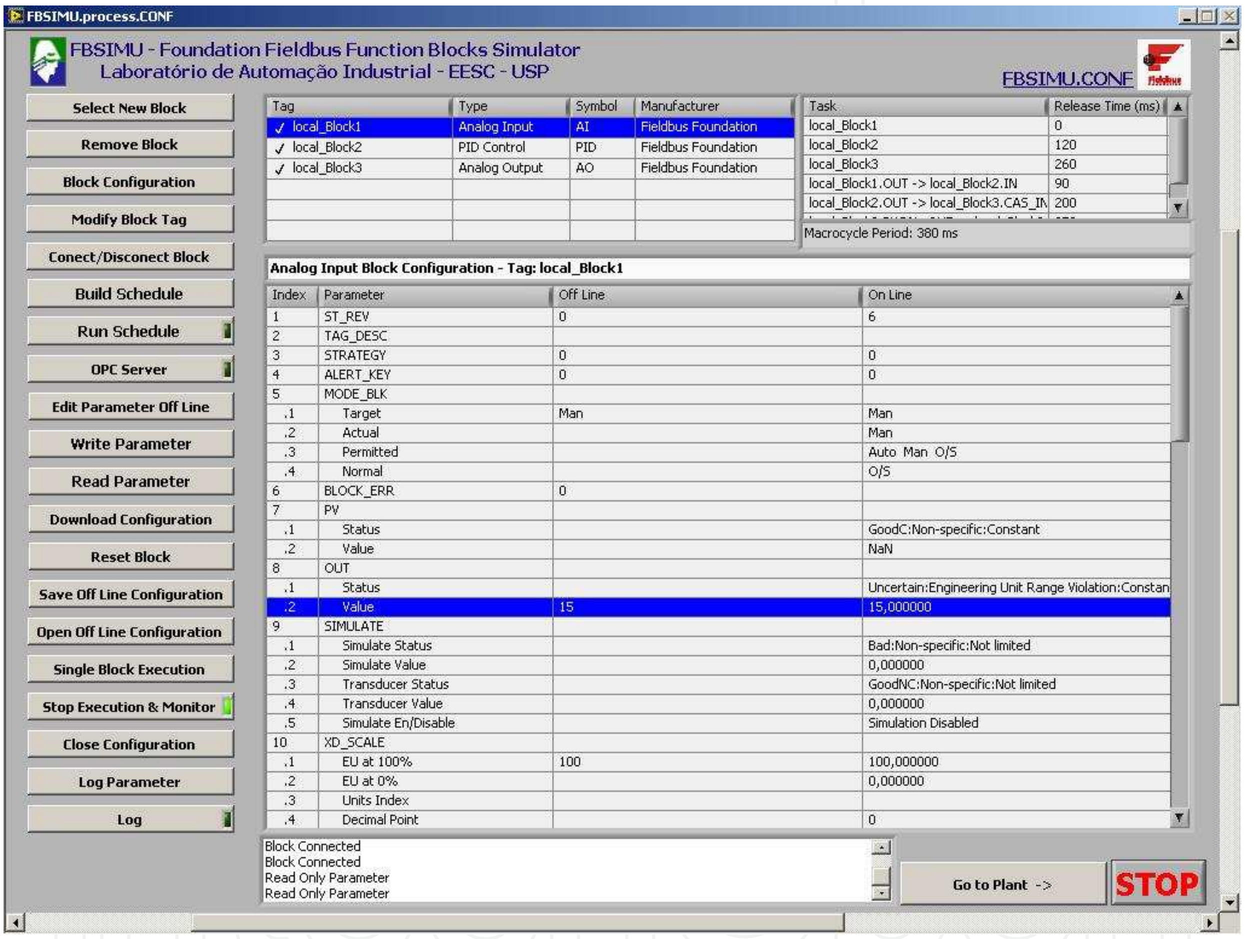
Each FB module is built in two different versions that share the same FB core: stand-alone and process. The stand-alone FBs are executed by user commands and controlled by its graphical user interface. Its execution can be performed independently of any other module, so the user is able to test the FB and simulate its operation under a controlled condition of inputs and outputs. The graphical user interface is intuitive and enables the user to execute the FB continually or in a step-by-step mode. The process version of a FB, on the other hand, is controlled remotely likewise real FBs. Each process FB has a unique identification and its operation is controlled by the user through the following commands:

- FB_Read: this service allows the value associated with a block parameter to be read.
- FB_Write: this confirmed service allows the value associated with a block parameter to be written.
- FB_Exec: this service triggers the block algorithm to be executed.

[3] LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench), a platform and development environment for a visual programming language (National Instruments)

- FB_Reset: this service allows default values associated with all block parameters to be written.

Process FBs do not have graphical user interface, they are instantiated by the FBSIMU.CONF in each simulation process. The communications between process FBs and the FBSIMU.CONF are performed programmatically and dynamically by the LabVIEW function ''Call by Reference Node''. It is important to note that the industrial transmitters are not considered in the FBSIMU architecture, i.e., function blocks are instantiated on the simulation without being allocated in specific ''virtual'' transmitters. The FBSIMU.CONF module graphical user interface for fieldbus configuration is shown in Figure 1.



FBSIMU - Foundation Fieldbus Function Blocks Simulator
Laboratório de Automação Industrial - EESC - USP

FBSIMU.CONF

Tag	Type	Symbol	Manufacturer
✓ local_Block1	Analog Input	AI	Fieldbus Foundation
✓ local_Block2	PID Control	PID	Fieldbus Foundation
✓ local_Block3	Analog Output	AO	Fieldbus Foundation

Task	Release Time (ms)
local_Block1	0
local_Block2	120
local_Block3	260
local_Block1.OUT -> local_Block2.IN	90
local_Block2.OUT -> local_Block3.CAS_IN	200

Macrocycle Period: 380 ms

Analog Input Block Configuration - Tag: local_Block1

Index	Parameter	Off Line	On Line
1	ST_REV	0	6
2	TAG_DESC		
3	STRATEGY	0	0
4	ALERT_KEY	0	0
5	MODE_BLK		
.1	Target	Man	Man
.2	Actual		Man
.3	Permitted		Auto Man O/S
.4	Normal		O/S
6	BLOCK_ERR	0	
7	PV		
.1	Status		GoodC:Non-specific:Constant
.2	Value		NaN
8	OUT		
.1	Status		Uncertain:Engineering Unit Range Violation:Constan
.2	Value	15	15,000000
9	SIMULATE		
.1	Simulate Status		Bad:Non-specific:Not limited
.2	Simulate Value		0,000000
.3	Transducer Status		GoodNC:Non-specific:Not limited
.4	Transducer Value		0,000000
.5	Simulate En/Disable		Simulation Disabled
10	XD_SCALE		
.1	EU at 100%	100	100,000000
.2	EU at 0%		0,000000
.3	Units Index		
.4	Decimal Point		0

```
Block Connected
Block Connected
Read Only Parameter
Read Only Parameter
```

Fig. 1. FBSIMU.CONF graphical user interface for fieldbus configuration

4.2 Physical plant simulations

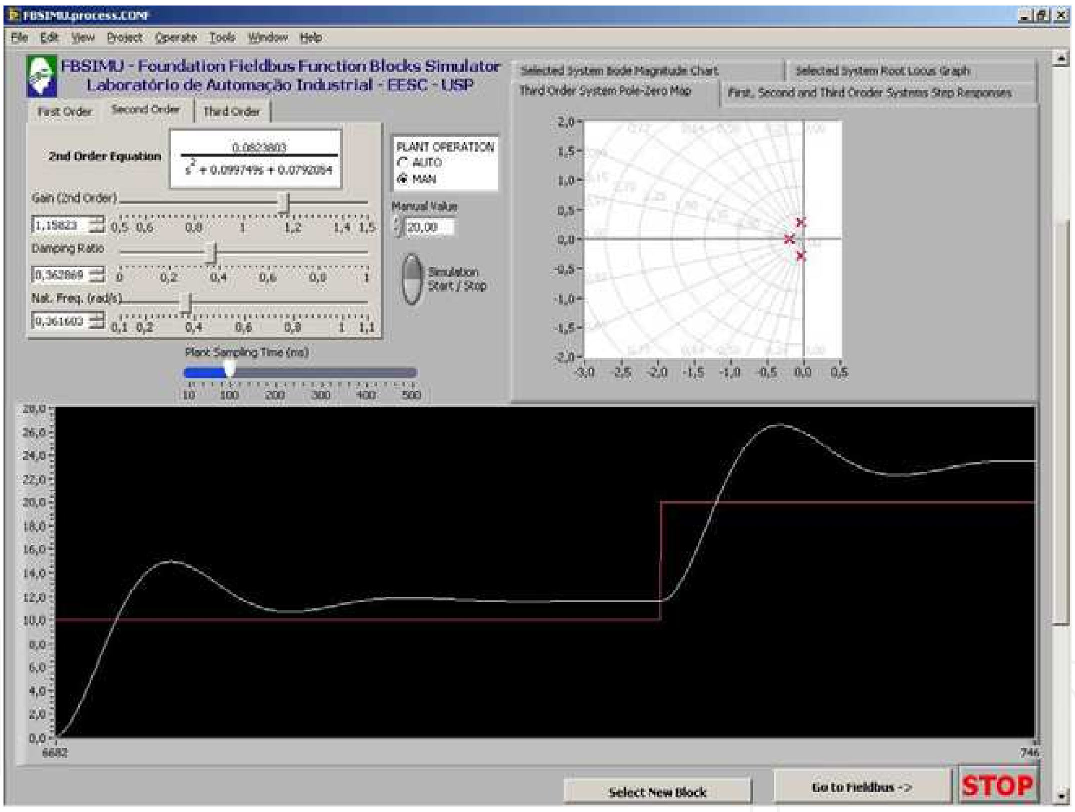
The plant module cyclically executes a discrete single variable (SISO) linear ARX (Auto-Regressive with Exogenous Inputs) mathematical structure (Ljung, 1999). This module is configured on the FBSIMU.CONF and simulates the controlled plant. The adopted ARX structure is represented by Equation 1, where k is the discrete time instant, Y is the output vector, U is the input vector, i is the number of MIMO plant inputs and outputs, na is the number of output regressors, and nb is the number of input regressors. In the current version, i is set to 1 (one) to reflect a SISO model.

The simulated plant dynamic behavior is modeled on the dynamic matrixes A and B. It must be observed that the number of regressors limits the model dynamic order and that all regressors must be initialized prior to starting the simulation.

$$Y_{ix1}(k) = \sum_{s=1}^{na} As_{ixi} \otimes Y_{ix1}(k-s) + \sum_{s=1}^{nb} Bs_{ixi} \otimes U_{ix1}(k-s) \qquad (1)$$

As the user chooses the plant order (1st, 2nd or 3rd) and dynamics (gain for 1st and 2nd order systems, damping ratio, natural frequency and time constant), the selected plants' Bode Magnitude Chart, Pole-Zero Map, Root Locus Graph and the Step Response are instantly presented on the graphical user interface.

A white noise generator function adds a simulated acquisition noise to each plant output bounded by user configurable amplitude. Figure 2 shows the FBSIMU.CONF module graphical user interface for plant configuration.



2nd Order Equation $\frac{0.0823803}{s^2 + 0.099749s + 0.0792054}$

Gain (2nd Order) 1,15823

Damping Ratio 0,362869

Nat. Freq. (rad/s) 0,361603

PLANT OPERATION
○ AUTO
● MAN

Manual Value 20,00

Fig. 2. FBSIMU.CONF graphical user interface for plant configuration

4.3 Simulation Architecture

The proposed execution model for the fieldbus simulation on FBSIMU is considered hybrid, because some tasks are event-driven while other tasks are time-triggered, according to table 1. All tasks related to the user interface are event-driven, they are executed after a user action such as selecting a new block, configuring schedule table, saving a configuration or starting the execution.

On the other hand, the tasks related to executing FBs according to a schedule table, plant simulation, and online monitoring of FBs are time-triggered.

Module	Priority	Execution	Timeout	Determinism
GUI & User commands	Low-Low	Event driven	1 sec.	No
FB Schedule	High-High	Time triggered according to the schedule table	No	Yes
Plant Execution	High	Periodic with configurable period	No	Yes
Online FB Parameters Monitoring	Low	Periodic with period = 500ms	No	Yes

Table 1. FBSIMU task set

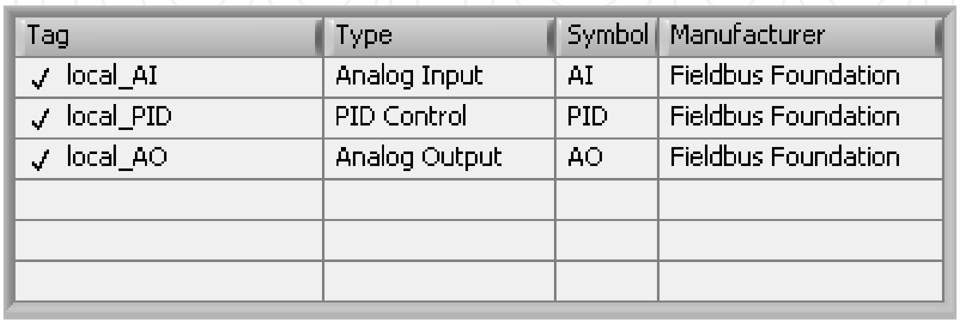
Once all tasks are performed on a single microprocessor they are, naturally, concurrent. The proposed solution for preventing unexpected delays of time-triggered tasks (considered critical) due to executing event-driven tasks (considered non-critical) is adopting priority levels for each task and preemptive execution mode.

In the preemptive execution mode, a higher priority task that is ready to execute preempts all lower priority tasks, which are also ready to execute or actually during execution. Table 1 summarizes the FBSIMU task set and its timing and execution characteristics.

4.4 A typical simulation experiment

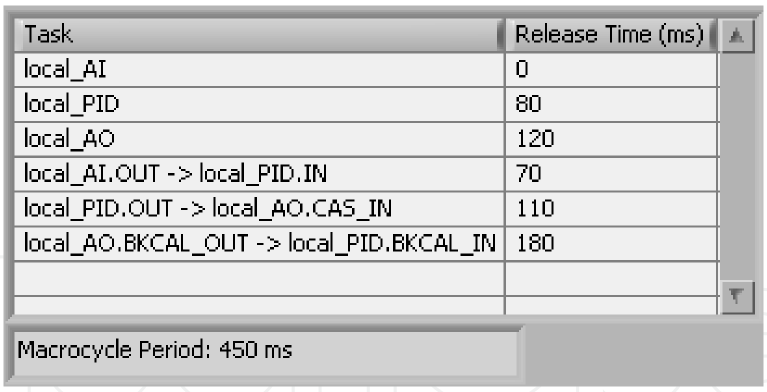
The intrinsic flexibility of simulation tools opens a wide range of FBSIMU experiments where users can exploit the effect of important communication parameters and configurations found in industrial FF systems. Practical experiments consist in comparing given simulated fieldbus system performances over different operation conditions. The results can be analyzed via log files or graphically on charts.

For typical simulation sessions, a specific FB control strategy should be defined. Then, the period of the macrocycle must be set in milliseconds, and all the release times (in milliseconds) of each FB execution and FB link must be defined regarding the macrocycle start instant. Figures 3 and 4 present an example of these configurations on the FBSIMU.



Tag	Type	Symbol	Manufacturer
✓ local_AI	Analog Input	AI	Fieldbus Foundation
✓ local_PID	PID Control	PID	Fieldbus Foundation
✓ local_AO	Analog Output	AO	Fieldbus Foundation

Fig. 3. FBSIMU block list

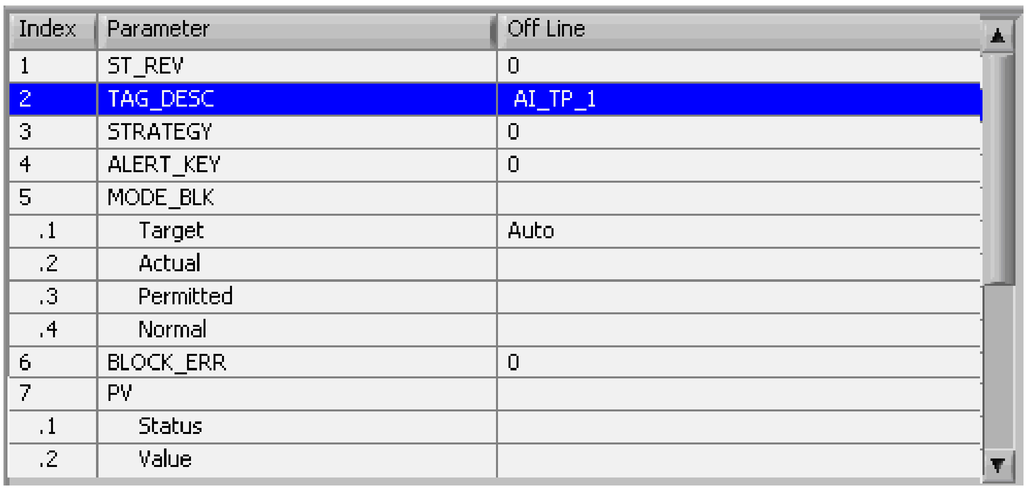


Task	Release Time (ms)
local_AI	0
local_PID	80
local_AO	120
local_AI.OUT -> local_PID.IN	70
local_PID.OUT -> local_AO.CAS_IN	110
local_AO.BKCAL_OUT -> local_PID.BKCAL_IN	180

Macrocycle Period: 450 ms

Fig. 4. – FBSIMU schedule table

The configuration parameters from all FBs on the schedule table must be set in a parameter table, as shown in Figure 5, to support the proposed strategy (for example, Block Mode, Scaling, Gains), exactly likewise a real block strategy configuration.



Index	Parameter	Off Line
1	ST_REV	0
2	TAG_DESC	AI_TP_1
3	STRATEGY	0
4	ALERT_KEY	0
5	MODE_BLK	
.1	Target	Auto
.2	Actual	
.3	Permitted	
.4	Normal	
6	BLOCK_ERR	0
7	PV	
.1	Status	
.2	Value	

Fig. 5. FB parameter table

The last step is to link the input and output parameters from the AI and AO blocks to the plant simulation module as represented in Figure 6. The connection between an Analog Output block (AO) and the plant input (manipulated variable - MV) and the connection between the plant output (primary value - PV) and the Analog Input block (AI) are configured by the user for close loop experiments. Alternatively, the user may connect only the plant PV to the AI block for an open loop simulation, or manually load the plant PV with a given numeric value.

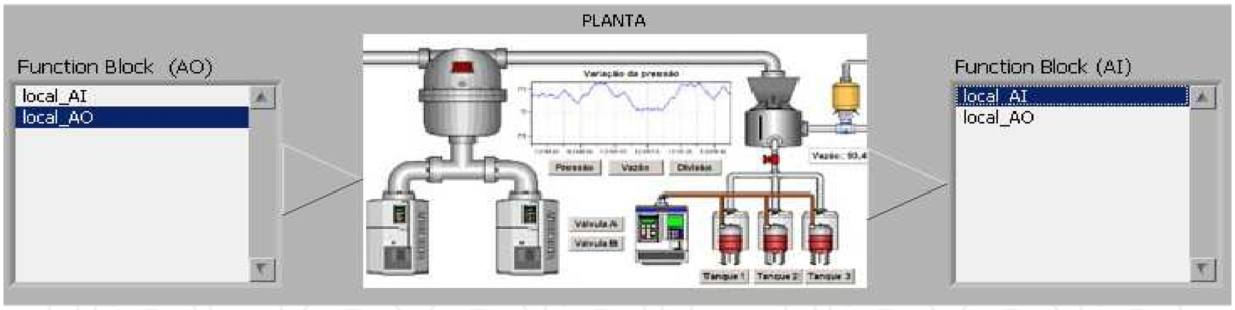


Fig. 6. - Plant simulation module

Finally, the user downloads the configuration to each FB and starts executing the schedule. During the execution, it is possible to monitor the parameter table with online parameter values and register the parameters on text files for further analysis. With the FBSIMU architecture, the FF operation scenarios can be configurable and different sequences of practice training can be defined to embrace fundamental concepts of fieldbus control systems, as well as practical situations of alarms or events handling. This characteristic is considered important because most of the traditional pilot plants equipped with fieldbus instrumentation offer just one or a few scenarios where a full sequence of practice experiments should be based on. Thus, the use of a simulated fieldbus system enables a flexible evaluation of the contribution and effect of the communication protocol on the overall system dynamics, which is an impossible goal considering that, in pilot plants equipped with real instrumentation, most communication configurations are fixed and, in most cases, inaccessible to end-users.

5. Multiagent Architecture

There is a number of requirements that ensures control of the production process. These include process variables, that is, data collected by the sensors that are often used for actuator actions. An incorrect interpretation and analysis of current data can result in the malfunctioning of the productive process. Thus, the functioning of a process can occur by combining these two items: collected data (sensors) and actions (actuators). Accordingly, we propose multiagent architecture which enables the analysis, interpretation and correction of data and events occurring in the fieldbus to improve production processes at the fieldbus level.

This architecture is composed of a multiagent system that generates inspection routines of the collected data in the plant's sensors. The aim is to induce the agents to evaluate field device data and investigate inconsistencies that may impede the productive process, such as lack of precision, external noise, alarm interpretation etc. During the process, the agents start functioning and perform tasks such as analyzing and correcting events through intelligent algorithm, as shown in Section 3.

Figure 7 shows that our architecture is composed of observation, diagnostic, and execution agents. The Observation Agent (OA) is responsible for monitoring field devices and checking for inconsistencies and faults. The Diagnostic Agent (DA) attempts to identify the type of fault (detected by OA) occurring in the field devices. Once the problem is diagnosed, the Execution Agent (EA) tries to correct it by implementing an intelligent algorithm. Another component in this architecture is a layer (LABVIEW/FIPA Layer) that allows agents to interact with the Foundation Fieldbus model. We use LABVIEW framework to develop based-agents FIPA (Polaków & Metzger, 2007). The main reason for using FIPA-compliant agents is their capacity to aggregate other FIPA agents in the architecture. We also use an OPC (OLE for Process Control) interface, such as that used by Seilonen et al. (2002b) to integrate agents with the fieldbus. In our study, we change the function block connections to perform a desired control algorithm and to make these function block interconnections act as agents.

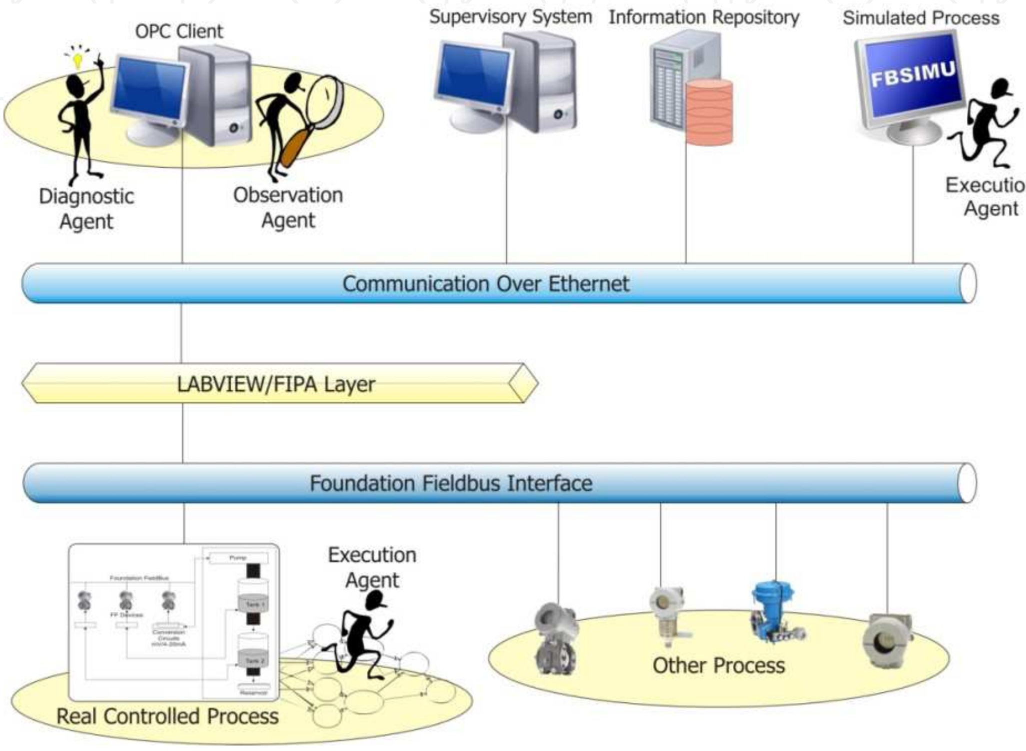


Fig. 7. – Proposed Multiagent architecture environment

The LABVIEW-FIPA layer allows communication with field devices through the OPC. Agents can access field devices and allocate function blocks. Agent learning occurs at higher levels (supervisory) which communicate with field devices (through the LABVIEW-FIPA/OPC layer). The gateway is responsible (physically) for establishing this communication. What is learned by the agent is stored in the Information Repository. This information is useful to other agents and reused in similar situations. Learning is implemented in computer supervisory machines as part of the agent. The agent action is performed at fieldbus levels, i.e., device function blocks (FB). The FB interconnections form an algorithm which controls a process. In our agent, this algorithm (ANN) is used in some instances to monitor devices (OA), and in others to correct faults (EA).

One of the OPC client and Foundation Fieldbus restrictions is the inability to allocate and deallocate function blocks in execution time. This operation is conducted by supervisors in plant control planning and any modification discontinues its operation. The operator performs the new configuration using proper software such as Syscon . Our strategy is to create a macro FB configuration from which others can be derived. In other words, changing the interconnections between allocated function blocks, it also changes control strategies. Figure 8 shows a number of possibilities of function block changes caused by a predetermined macro allocation. The advantage of this approach is the use of more than one ANNs. Agent actions are performed by an artificial neural network. A change in the interconnections between function blocks also leads to a change in ANN structure, which, in turn, changes agent structure. Thus, we exchange fieldbus agents through the function block interconnection configuration.

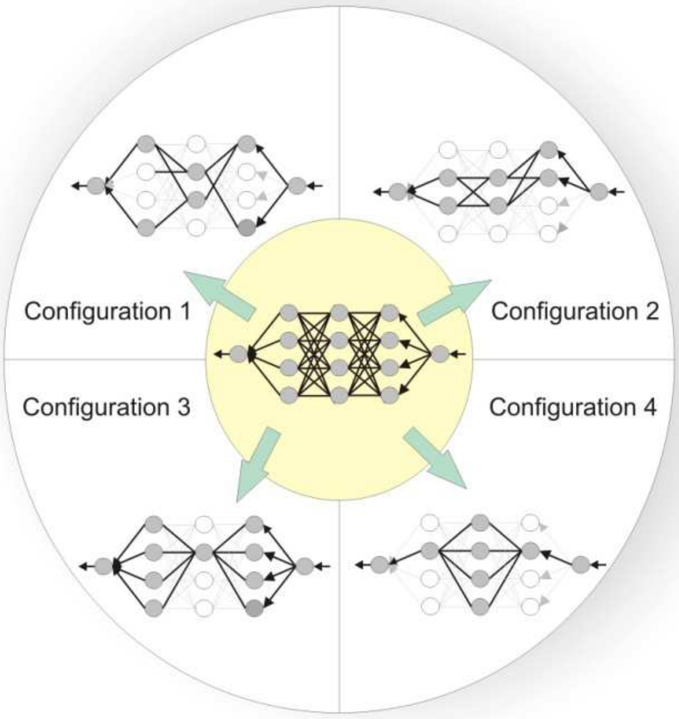


Fig. 8. - Change in Function Block structure

We have also apllied our architecture in FBSIMU to compare a real approach to a simulated one. It is important to underline that, in the FBSIMU, is possible to change all function blocks allocation in execution time, what is not possible in real fieldbus environment.

As an ANN, the agents go through two phases. The agents must learn (ANN train) and act (trained ANN use). They learn about the device (e.g. the OA learns how to predict device output values and the EA learns how to compensate for noise in device measurement). This training is conducted at the supervisory level and the device communicates with the fieldbus through a LABVIEW-FIELDBUS layer. The learned data (neural network weights) are stored in an information repository (IR).

In the learning phase the agents will train an artificial neural network to learn about fieldbus behavior. In the execution phase, the observation agents are able to monitor the field device and detect any faults therein. The OA and EA learns about the device or FBSIMU output (i.e., the OA learns how to predict device output values and the EA learns how to compensate for noise in device measurement, for example). The learned data (neural network weights) are stored in the information repository (IR). The EA uses this learned data to monitor devices. The system output is monitored In FBSIMU.

When a problem (malfunction) is detected, it must be correctly diagnosed to ensure a proper correction. The diagnostic agent (DA) consults the information repository to identify the type of problem that is occurring in the fieldbus. As soon as the problem is detected (by the observation agent), and identified (by the diagnostic agent) from the information repository, the execution agents decide the best configuration to resolve the problem. The execution agents are function blocks that is used in the devices. The organization of these blocks characterizes the way the EA solves the problem (algorithm). At the end, we have an error-free signal (or an output), for example. In the FBSIMU environment we replace the original schedule table for one which simulates an ANN function block configuration. A new table means a new process control. As previously discussed, the implementation of these agents is based on a structure formed by function blocks. Each function block executes a different kind of algorithm, and together they are capable of meeting a specific application, such as an ANN.

5.1 Observation Agents

Observation Agents, launched in a fieldbus, monitor the variable values of a number of devices. These agents aim to detect measurement anomalies in sensor values or actuator inaccuracies. Previous information about the system behavior is important to properly detect and diagnose faults. Thus, automation engineers can associate the faults with signal patterns. In recent years, research carried out in fault detection and isolation systems (FDI) has shown procedures that use computer intelligence procedures, such as the Fuzzy Logic system and Artificial Neural Networks.

In this work the observation agents use an ANN to predict the measured value of a device. Therefore, the OA must learn how to predict the measured signal behavior before it is launched in a fieldbus. This learning is accomplished at the supervisory level, that is, neural network training for a prediction problem. Thus, the observation agents know the expected signal behavior. When it is launched in fieldbus (ANN with trained weights), the OA tries to predict the next signal and compares it to the real signal. A difference in signals indicates a problem.

5.2 Diagnostic Agents

When a problem (fault) is detected, it must be correctly diagnosed to ensure proper correction. The diagnostic agent (DA), like an AO, uses neural networks to correctly diagnose which fault is occurring in the fieldbus. it is necessary identify which problem is occurring in the process. The DA is based on previous work (Fernandes et al., 2007), and it identifies the type of fault is occurring in the system. Like other agents, the DA is based on an ANN. It is implemented at the supervisory level which communicates with the fieldbus through the OPC client. The neural identification system is defined as a two-step identification process, signifying the existence of an ANN to evaluate a system output value. The general scheme of the functioning system is shown in Figure 9. In this case, while the level system is in execution, a system from the ANNs tries to find its identification using its ($x(k)$) inputs. Each time, output level system ($y(k)$) is compared to output identification system ($y'(k)$), generating a residue value ($r(k) = y(k) - y'(k)$) that is used later in the fault isolation/classification system. Then, one system analyzes the residue values and indicates the occurrence or not of faults. When faults are detected, it indicates which type is occurring. In Section 6, we show this approach applied in a real example.

5.3 Execution Agents

As soon as the problem is detected by the observation agents, the execution agents change function block interconnections to allocate an algorithm (ANN) which can fix the problem. The organization of these blocks determines how the EA solves the problem (algorithm). The execution agents can act (configure) in different ways to correct the errors. As it was previously discussed, the implementation of these agents is based on a structure formed by function blocks.

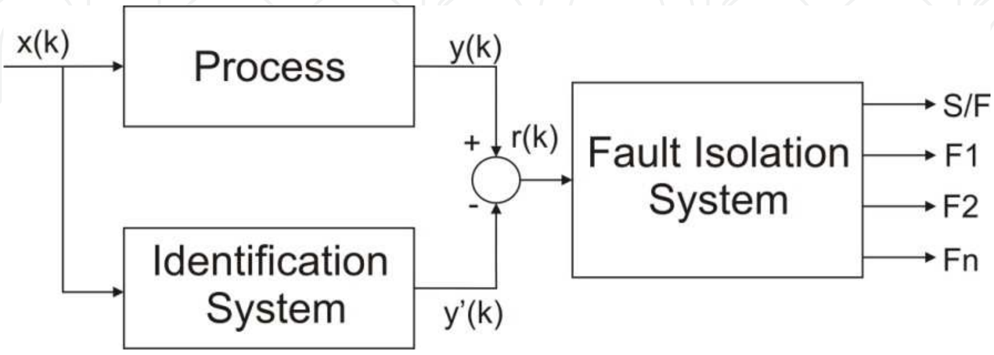


Fig. 9. - General scheme of FDI system (Fernandes et al., 2007)

A type of EA is illustrated in Figure 10. In this case, the structure formed by the function blocks contains a neural network-based noise filter. This algorithm (ANN) is able to remove noise from a measured signal. This structure is explained in Section 6.

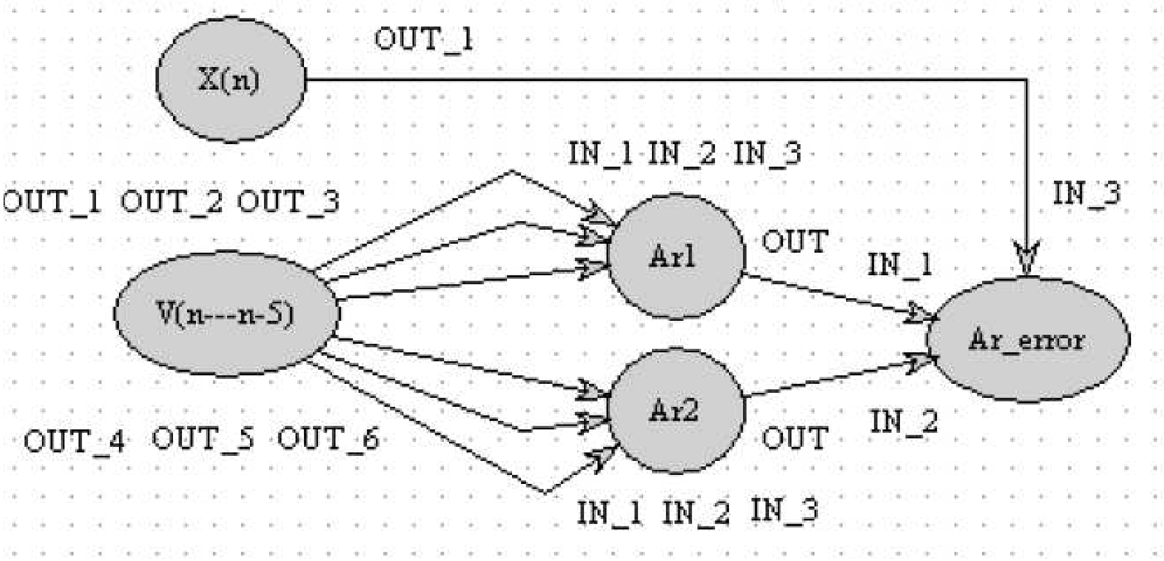


Fig. 10. - Noise Filter implemented as a Neural Network in Function Blocks as seen in SYSCON

The flexibility given by the application layer (represented by the function blocks) in the Foundation Fieldbus protocol enables different implementations at the field level. The combining of arithmetic and function characterizer blocks can produce a configuration similar to a neural network neuron (Silva et al., 2006). Thus, these examples show the variety of applications generated by different execution agents. For example: if the problem is noise interference collected by the sensor, the execution agents combine function blocks to form a neural network. With training values former acquired, this network can function as a filter (Costa et al., 2005), decreasing the value measured by the sensor. In the event of a tendency toward loss of accuracy in the measured values, detected by the observation agents, the EA can act as a prediction system, anticipating the presumed supervisory faults that may occur. The EA can also act as a recalibration algorithm in detecting a decalibrated sensor (Cagni et al., 2005).

6. MultiAgent Architecture Example

6.1 FBSIMU Example

In this example, we show the proposed multiagent architecture which uses the function blocks configuration exchange approach as it was previously showed. A simulated process automation was implemented for testing the feasibility of the architecture. The test environment consists of a simulated fieldbus-based automation system (FBSIMU) and a prototype agent application. The simulated process is controlled by a AI-PID-AO function blocks represented in Figure 4.

This environment is used in our example to detect and remove noise. First, the agents (ANN) undergo a learning process. The Observation Agent is trained to predict a FBSIMU output at given moment, based on its past outputs, considering that the simulated signal is noise free. The neural networks used by the Diagnostic Agent are trained to identify a kind of problem that may occur in the simulated environment. The EA is trained to act as a noise filter. The information acquired by the agent is stored in the Information Repository and can be used by other agents, in other situations, if necessary.

When the learning phase is over, the Observation Agent starts monitoring the FBSIMU output. Indeed, this agent is a prediction ANN. It monitors the output signal and predicts its corresponding value in the next step.

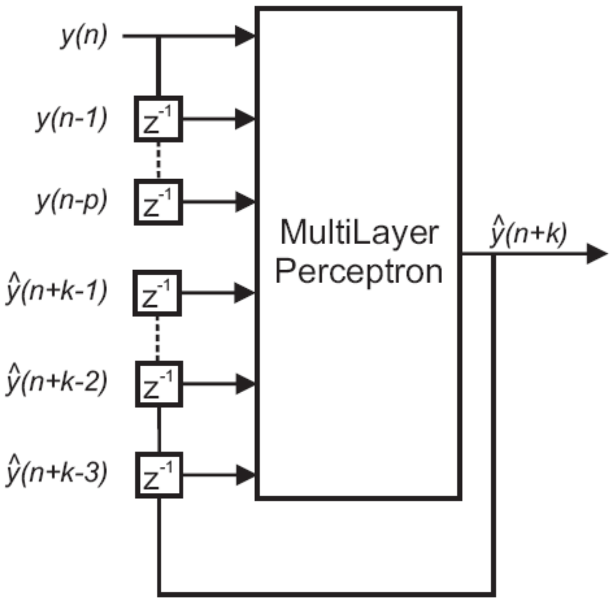
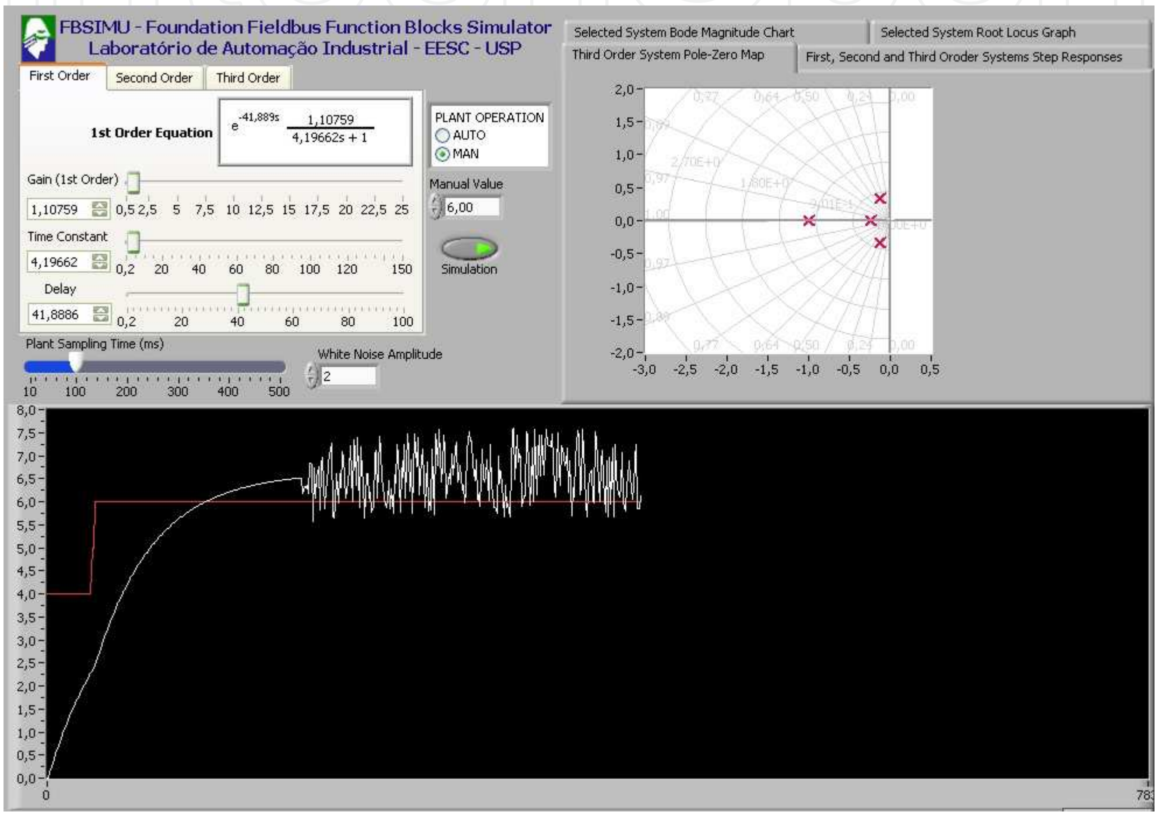


Fig. 11. - Recurrent Neural Network as Predictor

The type of prediction architecture used in this study is shown in Figure 11. It is a model with overall refeeding, resulting from a multiple-layer perceptron. The model has a single input, which is applied to delay line memory with *n*-units. It has a single output which is refed from the input of another delay line memory. The contents of these two memories are used to feed the input layer of the network.

In our test, FBSIMU introduces a simulated noise signal to the signal monitored by the agent (Figure 12). In certain moment, the output signal starts to exhibit different behavior as it was predicted. The difference between the measured and predicted signals is considered a problem by the OA.



FBSIMU - Foundation Fieldbus Function Blocks Simulator
Laboratório de Automação Industrial - EESC - USP

1st Order Equation $e^{-41,889s}\frac{1,10759}{4,19662s+1}$

PLANT OPERATION
○ AUTO
● MAN

Gain (1st Order) 1,10759

Time Constant 4,19662

Delay 41,8886

Manual Value 6,00

Plant Sampling Time (ms)

White Noise Amplitude 2

Fig. 12. – Output signal with noise in FBSIMU

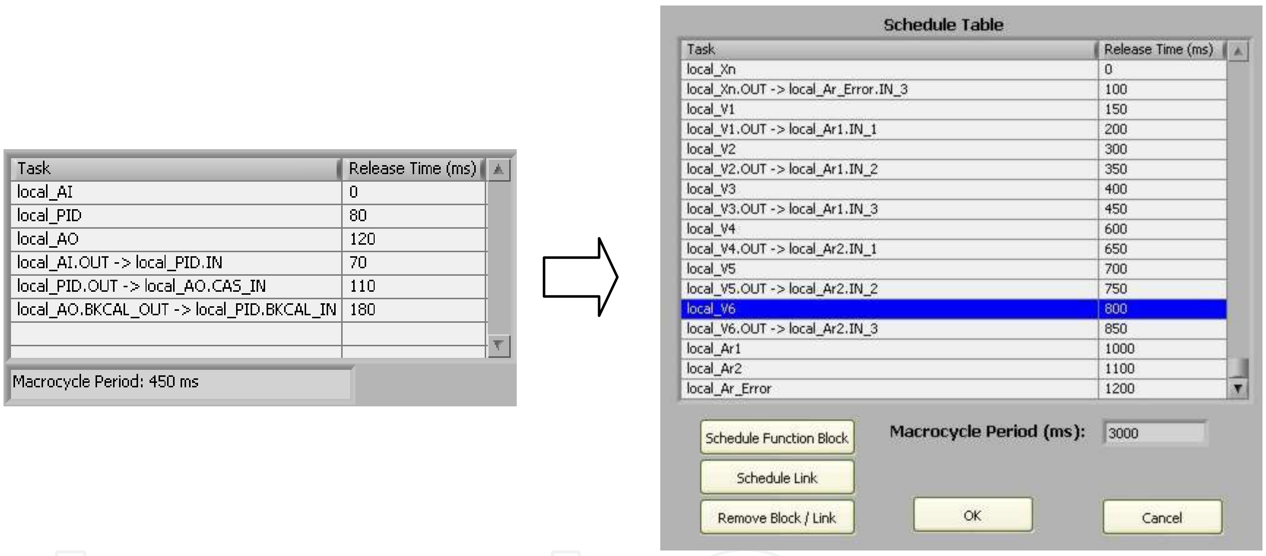
At this moment, it is necessary identify which problem is occurring in the process. The diagnostic agent (DA) is responsible for identifying the problem. Like other agents, the DA is based on an ANN. It is implemented at the supervisory level which communicates with the fieldbus through the OPC client.

As mentioned in Section 5.3, the DA tries to find its identification using its inputs. Each time, FBSIMU (y(k)) is compared to predicted output (y'(k)), provided by AO, generating a residue value ($r(k) = y(k) - y'(k)$). This residue value indicates the occurrence or not of faults. If it is closer to zero, this indicates no faults. Otherwise, it indicates which type is occurring. If residue is positive, it indicates a positive noise, if not, it indicates negative noise. In this particular example, we simplify the detection. The Diagnostic Agent determines noise presence or not.

In this example, the DA detects positive. This means there is noise in the FBSIMU output. At this moment, the function block parameters change to allocate the EA as an ANN acting as a noise filter, trained recursively until reasonable noise extraction is achieved. In the FBSIMU the EA exchanges the schedule table (Figure 13). The normal function block schedule is replaced by a new table which represents the EA allocation (trained ANN to remove noise). This function block allocation (EA) continues until the problem is solved (removing noise), then, the function block parameters (schedule table) change again to allocate the normal control and OA, and it restarts tracking the FBSIMU output.

6.2 Real Environment Example

In this second example, we show the proposed multiagent architecture which uses the function blocks configuration exchange approach, as it was previously showed. A prototype version of process automation was implemented for research purposes, and a laboratory test environment was used for testing the feasibility of the architecture. The test environment consisted of a simulated test, a fieldbus-based automation system and a prototype agent application. The test process contained parts that were similar to industrial processes. This environment was presented for the first time in our previous work (Machado et al., 2008a), and the results were showed in Machado et al. (2008b).



Task	Release Time (ms)
local_AI	0
local_PID	80
local_AO	120
local_AI.OUT -> local_PID.IN	70
local_PID.OUT -> local_AO.CAS_IN	110
local_AO.BKCAL_OUT -> local_PID.BKCAL_IN	180

Macrocycle Period: 450 ms

Schedule Table

Task	Release Time (ms)
local_Xn	0
local_Xn.OUT -> local_Ar_Error.IN_3	100
local_V1	150
local_V1.OUT -> local_Ar1.IN_1	200
local_V2	300
local_V2.OUT -> local_Ar1.IN_2	350
local_V3	400
local_V3.OUT -> local_Ar1.IN_3	450
local_V4	600
local_V4.OUT -> local_Ar2.IN_1	650
local_V5	700
local_V5.OUT -> local_Ar2.IN_2	750
local_V6	800
local_V6.OUT -> local_Ar2.IN_3	850
local_Ar1	1000
local_Ar2	1100
local_Ar_Error	1200

Macrocycle Period (ms): 3000

Fig. 13. – Schedule Table Exchange

As we can see in Figure 14, the plant level is composed of two cascading tanks. The water that flows out from the small hole of tank 1 falls into tank 2. This tank also has a small hole through which the water falls directly to the reservoir. A pump impels the water from the reservoir to tank 1. In each tank there is a Foundation Fieldbus pressure sensor, used to measure the corresponding levels connected to the Fieldbus network. Besides the pressure sensors, an FF/ loop of current from a 4 to 20 mA converter is used to send signals to the water pump. The industrial network Foundation Fieldbus is connected to a supervisory computer through Ethernet network interfaces. All the device configuration processes are carried out from this computer, and later supervised. This system transmits signals to the pump input to allow for water injection (or not) in tank 1 and to control the water level in both tanks. There is also a PC (OPC client) that sends a simulated noise signal (red dotted line) to a device.

Like previous FBSIMU example, this environment is used by agents to detect and remove noise. The Observation Agent is trained to predict a sensor output at given moment, based on its past outputs, considering that the signal is noise free. The neural networks used by the Diagnostic Agent is trained to identify a kind of problem that may occurs in the tanks. The EA is trained to act as a noise filter. When the learning phase is over, the Observation Agent starts monitoring the field device, which is allocated in the function blocks. This agent is allocated as a prediction ANN in field devices. It monitors the output signal and predicts its corresponding value in the next step. The ANN model for prediction is the same showed in Section 6.1 (Figure 11).
In our test, a PC (OPC client) sends a simulated noise signal to the device monitored by the agent. In Figure 15 A, we can observe both real and predicted signals. In half of samples, the sensor output signal starts to exhibit different behavior from that predicted (noise added by OPC Client). The difference between the compared signals is considered a problem by the OA. This difference can be seen in Figure 15 B.

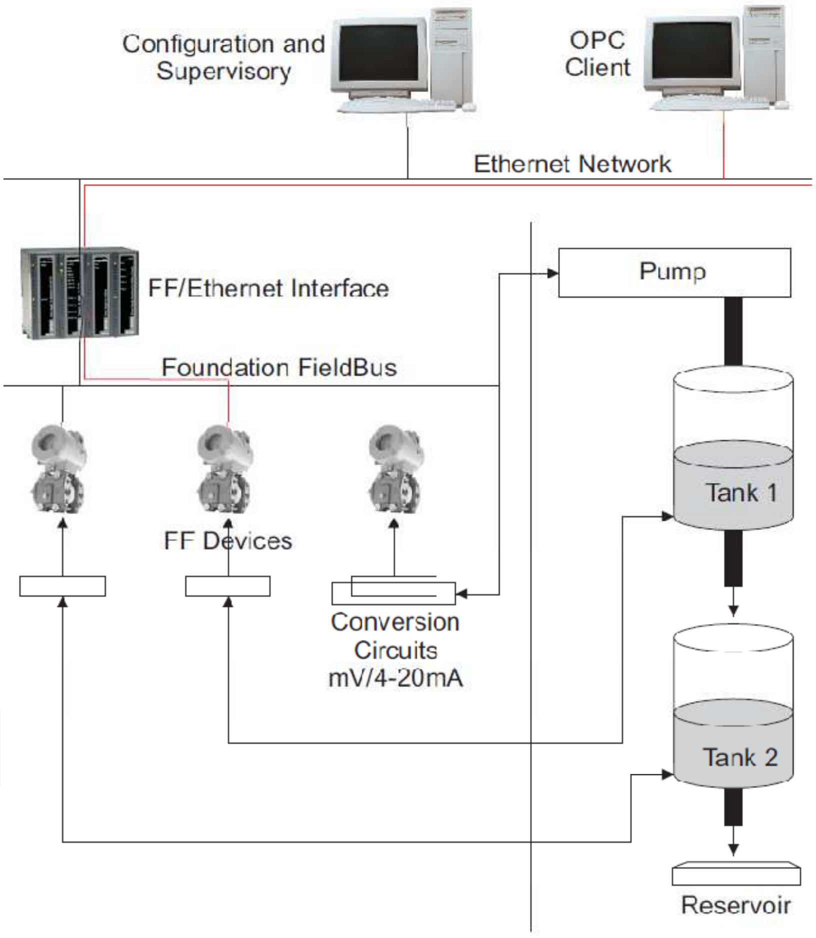


Fig. 14. – Laboratory environment

At this moment, the Diagnostic Agent (DA) is responsible for identifying a problem and selecting the best function block allocation to solve it. The DA is based on the previous described work (Section 6.1) which identifies what kind of fault is occurring in the tanks system. The DA is implemented in supervisory level and communicates with the fieldbus through an OPC client. The neural identification system was defined as identification in two steps, which means the existence of an ANN to evaluate the level of tank 1, and another to evaluate the level of tank 2.

With identification in two steps, it is possible to get two residues: r(1) and r(2), where r(1) = l(1) − l′(1) and r(2) = l(2) − l′(2). $l(1)$ is the real measured signal, and $l'(1)$ is the predicted signal. In this case, an ANN, named ANN 3, is trained by receiving as input data the values from $r(1)$ and $r(2)$. The networks output corresponds to a vector of $n+1$ numbers, numbers, where n is a quantity of faults that the network is able to classify.

Considering the two residues to detect and isolate the faults, the FDI system could detect in maximum 8 different faults, in which the situation $r(1) = 0$ and $r(2) = 0$ would be a normal behavior of the system. In view of the test, we managed to foresee only six types of faults with joint distinct residues.

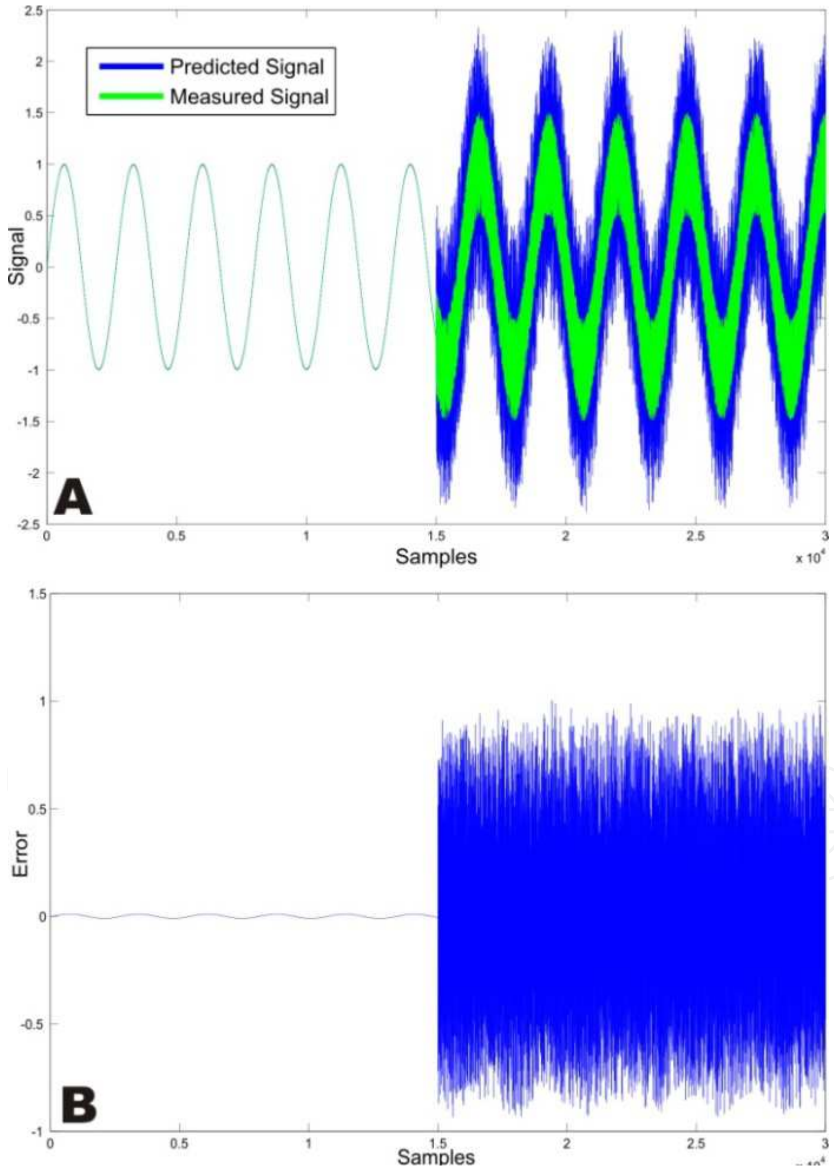


Fig. 15. – Real and Predict Signals

Table 2 shows the five types of faults selected for the result analysis, where (+) represents the positive residue, (-), the negative residue, and (0), the equal residue or very close to zero. In this example, the DA detects positive residue for $r(1)$ and null value for $r(2)$. This means there is a noise in tank 1 output sensor. At this moment, the function block parameters change to allocate the EA as an ANN, acting as a noise filter, once it was trained recursively until a reasonable noise extraction.

Fault	Description	R(1)	R(2)
	Absence of fault	0	0
1	New hole in tank 1. No fall of water in tank 2	-	0
2	Decrease of hole in tank 1	+	-
3	Decrease of hole in tank 2	-	+
4	Increase of hole in tank 2	0	-
5	Read error in Sensor 1. Positive Bias. (Noise Added)	+	0

Table 2 – Dispositions of the 6 fault residues

The EA (acting as a filter) acts immediately after noise detection. This function block allocation (EA) continues until the problem is solved, then, the function block parameters change again to allocate the OA and restart device tracking. It is important to emphasize that for experimental purposes the function block structure is replaced by another when the agents are exchanged (e.g., Observation Agents to Execution Agents). If the number of devices is substantial, the architecture can allocate many OA and EA at the same time, monitoring and acting on different devices and fault situations.

7. Conclusions

Nowadays, problems in the field devices are detected by supervisors through alarm triggers. From the proposed SMA architecture, the agents are able to detect and apply intelligent algorithms to solve these problems without user intervention. This article shows multiagent architecture which is able to detect and correct an undesired noise in a Foundation Fieldbus device and simulated environment (FBSIMU) by implementing function block intelligent algorithms. The intelligent algorithms use Artificial Neural Network (ANN) to find out about the noise and remove it. The agents encapsulate these ANNs and, using a LABVIEW-Fieldbus layer, can directely interact with field devices through an OPC client. In this approach the algorithm is implemented in the device function blocks providing a solution at fieldbus level.

This chapter provides two innovations in fieldbus research mentioned in our previous works. First, we have a dynamic function block interconnection exchange that allows the allocation of different neural network structures at fieldbus level. Accordingly, we have several control strategies (agents) allocated in field devices, which can monitor, detect, and correct a number of faults. The second innovation is the reusing of function block configurations. The same ANN structure can be used in different situations. That is, the same agent can act in other processes. The use of Information Repository allows us to share the ANN structure with other agents.

However, the main novelty in this chapter is the use of our approach in a function block simulated environment (FBSIMU). Thus, we can compare two types of function block intelligent algorithm implementation. The first one, in FBSIMU, is able to exchange all function blocks configuration in execution time. This approach is not permitted in real fieldbus environment. In this (second) case, we opt for a function blocks interconnections exchange. So, we can observe, despite second approach is well suited for a real Foundation Fieldbus environment, the full function block allocation and deallocation is more appropriate for exchange control algorithms in fieldbus devices function blocks. For safety reasons and respecting the industries patterns, the real time function block exchange was not implemented by foundation fieldbus manufacturers. We believe that our approach can supply this gap.

8. References

Albert , M.; Längle, T.; Wörn, H. (2003). Generic Diagnostic Functionalities Encapsulated within a Software Agent. *IAR-ICD/IFATIS/ MAGIC Workshop on Advanced Control and Diagnosis*.

Brandão, D. (2005). Ferramenta de simulação para projeto, avaliação e ensino de redes fieldbus, *Doctorate thesis*. Escola de Engenharia de São Carlos, USP.

Brennan, R. W.; Zhang, X.; Xu, Y.; Norrie, D. H. (2002). A reconfigurable concurrent function block model and its implementation in real-time java, Integr. *Comput.-Aided Eng.*, vol. 9, no. 3, pp. 263–279, 2002.

Cagni, E.; Pereira, D.; Pereira, A.; Doria Neto, A. D.; de Melo, J. D.; Guedes, L. A. (2005).The implementation of the self-calibration, self-compensation and self-validation algorithms for foundation fieldbus sensors are presented using standard function blocks, IEEE International *Conference on Computational Intelligence for Measurement Systems and Applications*, pp. 220–225.

Cavalieri, S.; Di Stefano, A.; Mirabella O. (1993). Optimization of acyclic bandwidth allocation exploiting the priority mechanism in the fieldbus data link layer. *IEEE Transactions on Industrial Electronics*, pp. 297–306.

Chen, J.; Wang, Z.; Sun, Y. (2002). How to improve control system performance using FF function blocks. In: *IEEE international conference on control application*. Glasgow, Scotland, pp.1022-1026.

Costa, I.; Doria Neto, A. D.; de Melo, J. D.; de Oliveira J. (2005). Embedded FASTICA algorithm applied to the sensor noise extraction problem of foundation fieldbus network, Neural Networks, 2005. IJCNN *'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 4, pp. 2217–2221.

Fei-Yue, W.; Haitao, Z.; Yunfeng, A. (2005). An OSGi and agent based control system architecture for smart home, *Proc. Networking, Sensing and Control*, pp. 13–18.

Feng, Q.; Bratukin, A.; Treytl, A.; Sauter, T. (2007). A flexible multi-agent system architecture for plant automation, in *5th IEEE International Conference on Industrial Informatics (INDIN 2007) Industrial Informatics*, IEEE Transactions on, pp. 1047–1052.

Fernandes, R. G.; Silva, D. R.; Guedes, L. A.; Doria Neto, A. D. (2007). An implementation of a fault detection and isolation system on foundation fieldbus environment. *International Journal of Factory Automation*, Robotics and Soft Computing, vol. 3, pp. 130–136.

Ferreiro, R.; Vidal J.; Pardo, C.; Coego, J. (1997) Fieldbus: Preliminary design approach to optimal network management. In: *IEEE international workshop on factory communication systems*, pp. 321 – 325.

Fieldbus Foundation. (1999b). *Foundation specification function block application process,* Part 3: FF-892 – FS1.4. Austin, USA.

Fieldbus Foundation. (1999c). *Foundation specification function block application process*, Part 5: FF-894 – DPS0.95. Austin, USA.

Fieldbus Foundation. FF-890-1.3. (1999a). *Foundation specification function block application process*, Part 1. Austin, USA.

Haykin, S. (1999). Neural Networks - A Comprehensive Foundation. *Prentice Hall*.

Hong, S.H.; Ko, S.J. (2001) A simulation study on the performance analysis of the data link layer of IEC/ISA fieldbus. SIMULATION, pp. 109–18.

International Electrotechnical Comission. (2000). IEC 61158: *Digital data communications for measurement and control – fieldbus for use in industrial control systems.* Switzerland. CD- ROM.

International Electrotechnical Comission. (2003). IEC 61784: *Digital data communications for measurement and control - Part 1: Profile sets for continuous and discrete manufacturing relative to fieldbus use in industrial control systems.* Switzerland. CD-ROM. 2003.

International organization for standardization. (2009). ISO/IEC 7498-1: Information technology – open systems interconnection – basic referencemodel: The basic model. Switzerland. CD-ROM.

Jennings, N.R.; Bussmann, S. (2003) Agent-based control systems: Why are they suited to engineering complex systems?, in *IEEE Control Systems Magazine*, v. 23, Issue 3, pp. 61-73.

Ljung, L. (1999). *System identification- theory for the user*. Englewood Cliffs: Prentice Hall.

Machado, V.; Doria Neto, A. D.; de Melo, J. D.; Ramalho, L.; Medeiros, J. (2008a). Multiagent architecture for function blocks: Intelligent configuration strategies allocation, in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, 2008, pp. 1377–1382.

Machado, V.; Doria Neto, A. D.; de Melo, J. D.; Ramalho, L.; Medeiros, J. (2008b). A neural network multiagent architecture applied to fieldbus intelligent control. in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, pp. 567–574.

Petalidis, N.; Gill, D. S. (1998) The formal specification of the fieldbus foundation link scheduler in E-LOTOS. In: *International conference on formal engineering methods*.

Pinotti Jr., M.; Brandão, D. (2005). A flexible fieldbus simulation platform for distributed control systems laboratory courses. *The International Journal of Engineering Education*, pp. 21(6):1050–8. Dublin.

Pirttioja, T.; Pakonen, A.; Seilonen, I.; Halme, A.; Koskinen, K. (2005). Multi-agent based information access services for condition monitoring in process automation, in *Proc. INDIN '05, 3rd IEEE International Conference on Industrial Informatics*, pp. 240 – 245.

Polaków, G.; and Metzger, M. (2007). Agent-Based Approach for LabVIEW Developed Distributed Control Systems. *In Proceedings of the 1st KES international Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. N. T. Nguyen, A. Grzech, R. J. Howlett, and L. C. Jain, Eds. Lecture Notes In Artificial Intelligence, vol. 4496. Springer-Verlag, Berlin, Heidelberg, pp. 21-30.

Pop, T.; Eles, P.; Peng, Z. (2002). Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In: *10th international symposium on Hardware/software* codesign. pp. 187 – 192.

Russell, S.; Norvig, P. (2003) Artificial Intelligence: A Modern Approach, 2nd ed. *prenticeort: prentice*.

Schoop, R.; Colombo, A.W.; Suessmann, B.; Neubert, R. (2002). Industrial experiences, trends and future requirements on agent-based intelligent automation, in Proc. IECON 02, *28th IEEE Annual Conference of the Industrial Electronics Society*, pp. 2978–2983, vol.4.

Seilonen, I. (2006) An extended process automation system: An approach based on a multi-agent system. *Ph.D. dissertation*, Helsinki University of Technology, Espoo, Finland.

Seilonen, I.; Appelqvist, P.; Koskinen, K. (2002a). Agent-based approach for faulttolerance in process automation systems, in *Proceedings of the 3rd International Symposium on Robotics and Automation* (ISRA 2002).

Seilonen, I.; Pirttioja T.; Appelqvist, P. (2002b). Agent technology and process automation, pp. 31–35.

Silva, D.; Guedes L. A.; Doria Neto, A. D.; de Melo, J. D. (2006) "Neural networks implementation in foundation fieldbus environment: A case study in neural control", *International Journal of Factory Automation, Robotics and Soft Computing,* pp. 48–54.

Taylor, J. H.; Sayda A. (2005). An Intelligent Architecture for Integrated Control and Asset Management for Industrial Processes, in *Proc. IEEE International Symposium on Intelligent Control*, Limassol, Cyprus, pp. 27-29.

Theiss, S.; Vasyutynskyy, V.; Kabitzsch, K., (2008). AMES - a resource-efficient platform for industrial agents, *Factory Communication Systems, 2008. WFCS 2008. IEEE International Workshop on* , pp.405-413, 21-23.

Wang, Z.; Yue, Z.; Chen, J.; Song, Y.; Sun, Y. Realtime characteristic of FF like centralized control fieldbus and it's state-of-art. In: IEEE international symposium on industrial electronics. 2002.

Weyns, D.; Holvoet T. (2007). Architectural design of a situated multiagent system for controlling automatic guided vehicles, *International Journal on Agent Oriented Software Engineering*, vol. 1, no. 4, pp. 1–39.

Weyns, D.; Schelfthout, K.; Holvoet, T.; Lefever, T.; (2005). "Decentralized control of e'gv transportation systems," in Autonomous Agents and Multiagent Systems, pp. 67–74. [Online]. Available: citeseer.ist.psu.edu/weyns05decentralized.html

Factory Automation

Edited by Javier Silvestre-Blanes

ISBN 978-953-307-024-7
Hard cover, 602 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

Factory automation has evolved significantly in the last few decades, and is today a complex, interdisciplinary, scientific area. In this book a selection of papers on topics related to factory automation is presented, covering a broad spectrum, so that the reader may become familiar with the various fields, and also study them in more depth where required. Within various chapters in this book, special attention is given to distributed applications and their use of networks, since it is one of the most relevant subjects in the evolution of factory automation. Different Medium Access Control and networks are analyzed, while Ethernet and Wireless networks are looked at in more detail, since they are among the hottest topics in recent research. Another important subject is everything concerning the increase in the complexity of factory automation, and the need for flexibility and interoperability. Finally the use of multi-agent systems, advanced control, formal methods, or the application in this field of RFID, are additional examples of the ideas and disciplines that experts around the world have analyzed in their work.

Vinicius Ponte Machado, Dennis Brandao, Adriao Duarte Doria Neto and Jorge Dantas de Melo (2010). A Multiagent Architecture Based in aFoundation Fieldbus Network Function Blocks, Factory Automation, Javier Silvestre-Blanes (Ed.), ISBN: 978-953-307-024-7, InTech, Available from: http://www.intechopen.com/books/factory-automation/a-multiagent-architecture-based-in-afoundation-fieldbus-network-function-blocks

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821