Web System for the Remote Control and Execution of an IEC 61499 Application

Oana ROHAT, Dan POPESCU

Faculty of Automation and Computer Science, Politehnica University,
Splaiul Independenței 313, București, 060042, Romania,
oana.rohat@gmail.com; dan_popescu_2002@yahoo.com

Abstract: Remote control and execution gives the possibility of designing applications where any controller can have access to high computational power and knowledge just by use of standard network interfaces. This work describes the structure and components of a system that allows a user to select an algorithm from a web library, configure the execution parameters and remotely control a plant application that is implemented based on the IEC 61499 standard. The system is based on open technologies like php, Java and IEC 61499 function blocks allowing the easy integration and interoperability of its components. The system was successfully tested and represents a novel approach in developing flexible, efficient and reliable applications.

Keywords: web application; remote execution; distributed systems; function block programming; software integration.

1. Introduction

The design of flexible, efficient, reliable control systems is one of the main concerns in modern industrial applications. The solution is developing modular, Internet-enabled, reusable control logic systems, capable of working in distributed networks, and able to incorporate modern technologies like smartphones in the process control actions. A great attention was given over the last decades to improving the process control design and programming methods that provide patters or generic software objects and can be used in different applications with minimum reconfiguration and integration effort [1-2]. The use of such objects provide great help for process control engineers in designing higher quality and more efficient automation systems in a shorter period of time.

An important support in this direction came from the IEC 61499 standard [3]. It allows developing platform-independent, reusable and even self-configurable function blocks that give the possibility of exchanging information between interconnected controllers in a distributed network. Innovative applications were developed in domains like Smart GRIDs, Material Handling Units, Building Management Systems and Smart Factories to exploit the standard's benefits and prove its reliability [4].

Not much attention was paid until now to the web integration of an IEC 61499 application. The standard provides some standard communication objects [5] that can be further developed to implement industrial communication protocols like Modbus, Profibus, CAN, OPC etc.

Using the free FBDK tool for IEC 61499 application development and execution, this paper aims to further exploit the standard's benefits and develop a web-based system for the remote configuration and execution of a function block, with the results being sent to a specific device. This provides great flexibility and efficiency to the engineering process, as the distributed controller has access to greater computational power than its own resources. That way the system provides great accessibility and reliability in using a wide range of complex algorithms without the need of a controller upload and just by using an Ethernet communication link. Applications that are independent of the process time (like model learning, detecting optimal parameters, algorithm testing, plant risk analysis, image processing etc.) or the ones that don't have hard real-time constraints (like slow process control plants from the agriculture or biology domains, building management systems etc.) are the ones that can benefit most from the implementation based on such a remote execution system.

2. IEC 61499 and FBDK Support in Implementing Remote Web-Controlled Applications

One of the main paradigms behind the IEC 61499 standard refers to the development of truly distributed systems [6]. The standard

introduces the concept of system configuration that represents a set of interconnected devices with one or several resources that can execute an application or parts of it. This allows the process control engineer to imagine and test the functionality of an application as a whole and then "cut" it and assign parts of it to different devices that are involved in the control process and are connected in a distributed network.

Following this methodology, the remote control of an application means the separation of the command or user input functions and their implementation in a web structure. To lower the processing effort, only the execution parameters or application output functions are maintained at the controller level, while the algorithm implementing the desired functionality runs on a remote server. The standard provides all the needed support in designing the distributed application but no web-interface related functions are available. The integration with a web application is possible only by using a communication interface.

The IEC 61499 standard provides two communication objects that are also available in FBDK: Client/Server for bidirectional communications and Publish/Subscribe for unidirectional communication [7]. When using the Publish/Subscribe communication pattern the information flow is directed from the Publish to the Subscribe block. Several Subscribers can receive data from the same Publisher but no feedback is sent back for transmission acknowledgement or verification.

The link between the pairing blocks is done by an ID parameter that represents the multicast address. This parameter must have the same value on each side to make a connection. All parameters are transmitted from the sender to the receiver in the same order they were configured at the input of the function block.

These function blocks can also be used for web integration if they are used to implement open services communication protocols. They can be used for data transfer between remote devices but cannot be used for execution control. More advanced functions for OPC-UA protocol or web servers are available in commercial engineering software applications like NxtStudio but are not available in FBDK.

To fully control a distributed component of an application remotely, a user must also have the possibility to start or stop that function block. As is the case of most process control programming environments, the execution control of a system configuration or of a function block is not directly accessible in FBDK. As is later detailed in Section 3, FBDK has a "back door", the System Manager Configuration tool, which can be used for such operations.

3. System Modelling and Components Integration

The system for the remote execution of an IEC 61499 application was developed as a web library allowing access to different algorithms, organized according to their purpose and functionality. Using such a system implies little reconfiguration effort at the plant level which refers to changing the source of the corresponding controlled parameters. These parameters are usually entered manually, at the time of the application development or by use of a HMI. To run an algorithm, a user must add a communication interface that will link the desired control parameters to the remote execution system.

As can be seen in Figure 1, the system has three main entities: the web interface, controlled by the user, the library server which is independent of a particular application and the remote device whose particularities are not known. The user has access to the web interface that allows him to configure the parameters of the selected algorithm so that the execution results can be sent to the corresponding remote device from the plant application. He must add the corresponding communication interface at the plant level according to the whole system configuration. The communication was designed as unidirectional, so no feedback is received by the library from the remote device execution. Still, some cases (like the PID algorithm) may require a process parameter for algorithm execution. In this case, the execution will require an additional Subscribe/Publish pair so that the controlled value can be received from the plant. The user acts as an active observer able to adjust at runtime the algorithm parameters according to the process response.

The remote execution system is based on open technologies which allow the easy integration and interoperability of its components. Information exchange between different components is done using standard Internet

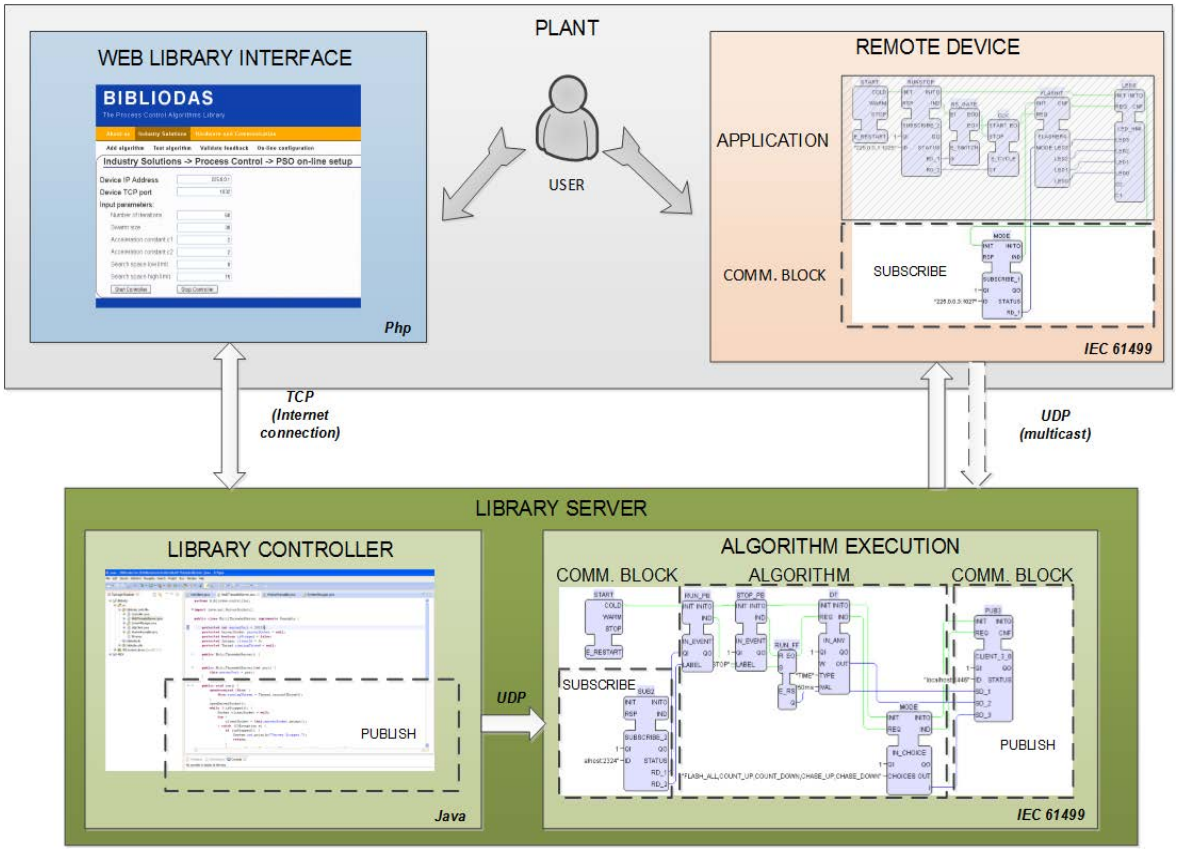

Figure 1. Integration of system components

communication protocols TCP and UDP because of their ease of implementation at each level.

The Web Library Interface was developed using HTML for interface design and PHP for database querying, forms and user session handling and data encapsulation into XML.

The Library Server has two components: the Library Controller and the Algorithm Execution environment. The Library Server is responsible for delivering the content for the web interface, for the management of execution commands and data flow through the system. The Library Controller was implemented using Java. The Algorithm Execution environment is based on FBDK.

The Remote Device implements a specific application using the IEC 61499 standard. The connection to this component is based on the Publish/Subscribe mechanism through UDP multicast packages. The details of the remote device are not known in the system, the only connection to it being through the communication interface.

4. How It Works

4.1 The Web interface

The Web Interface allows the user to select de desired algorithms based on their description of functionality, to configure the execution parameters and remote device address and to start or stop the execution of an algorithm. The execution parameters can be modified at runtime, so the user can adapt the algorithm according to the process response. Using PHP scripts, the Start Controller or Stop Controller commands are sent from the web page to the Library Controller using a TCP connection.

The Start Controller command that is sent from the web application encapsulates the algorithm parameters into an XML message that is transmitted to the Library Controller. This XML package contains information regarding the algorithm ID, the algorithm parameter names and corresponding values and the remote device IP address and port.

4.2 The library controller

The Library Controller is a Java application that is in charge of session management and parameter value transmission between the web application and the FBDK environment. When a user presses the Start Controller button the Library Controller executes the following actions: creates an instance of the algorithm, starts the specific algorithm execution and sends the parameters from the web interface to the algorithm function block. The controller can receive connections from many different web browser clients. These connections use the TCP protocol over a predefined port known by both web browser and the Library Controller.

Once the XML message from the Web application is received, the Library Controller parses it and extracts the parameters and their values. The parameter fields and their values are stored in a hash map for later use, when Library Controller will need to send these parameters through the Publish/Subscribe mechanism to the Algorithm Execution System.

4.3 Starting the Algorithm Execution System

Once a new instance of the algorithm is created, the Library controller needs to start its execution. Using the FKDB runtime libraries the controller makes a call to the "fb.rt.tools.SystemManager" Java application that will open a stand-alone FBDK system configuration. The System Manager library uses a .sys file received as a command line argument and starts the execution of all the block components of that system in separate panels.

In case of conventional use of the FBDK environment running the same algorithm twice simultaneously is not possible. That is because two instances of the same algorithm running on the same machine will create UDP port conflicts and the communication between publishers and subscribers will not be possible.

The System Manager is also responsible for starting the algorithm execution in the standard use of the FBDK environment. When the user selects an algorithm and clicks the "Run" button, the FBDK Editor calls the System Manager Java library with the corresponding .sys file to create an instance of that algorithm.

Sending the parameters from the Library Controller to the Algorithm Execution module is done using the Publish – Subscribe mechanism. The publisher sends multicast UDP packets on a specific UDP port over the network. For the subscriber blocks to receive this information, the FBDK environment will open a UDP socket on the same UDP port as the publishing blocks on which it will listen for incoming messages. For each algorithm that is running at a specific period of time, a different pair of communication blocks must be used. These blocks will also use different communication sockets.

In the current implementation, for the communication between the Java Controller and Algorithm System to allow different users to use the same algorithm, the IP/Port allocation must be different for each user session. If multiple users use the same algorithm, the session of the first user will use the default IP of 225.0.0.1 and UDP port 1024. The allocation of the UDP ports follows an incremental basis that adds the value of 1 for each new executed algorithm starting from 1024. Once the maximum number of 65535 is reached, the IP is then incremented and the counting restarts. In this way, different users will be able to use the same algorithm and the Java Controller will be able to send the parameters of the algorithm introduced by the user in the web page to the correct Algorithm System.

The FBDK logic which is responsible for the algorithm execution must use a subscribe block with a number of inputs according to the number of required algorithm parameters. As mentioned previously, this subscriber block will listen for incoming publisher packets on a specific UDP port.

The Library Java Controller, since it is not an FBDK application, must emulate an FBDK publisher block by creating a UDP socket on the corresponding UDP port. Then it must convert the information into FBDK custom packets that an FBDK subscriber block will know how to understand.

To achieve this, we have used the well-known network packet capture tool Wireshark to analyse the exchanged packets between an FBDK publisher and subscriber over the loopback network interface. Then we have inspected the UDP payload and extracted the required fields that need to be customized by

the Library Controller. What we have noticed is that the parameters of a frame device that are entered as input in the FBDK editor are sent by the publisher blocks in an encoded format. The encoding used by FBDK is a custom one in which the values of an input parameter are inserted in the UDP packet, one after another. The encoding is documented in the Annex E.3.3.2 of IEC 61499-1 and referenced by the Presentation Layer chapter of the Compliance Profile of FDBK. Using this encoding the Java Controller publishers can send the parameters from the webpage to the FBDK native subscribers.

4.4 Multiple concurrent connections

Another aspect of the application development is handling concurrent connections from different users or when a user starts more algorithms. For this function the Library Java controller creates one thread for each incoming TCP connection received from the Web interface module. The new thread reads the XML received from the Web interface, parses the parameters, sends them to the Algorithm Execution module and transmits a feedback to the web interface for the confirmation of starting the execution.

4.5 Handling Internet connection problems

Internet - Based Control Systems (IBCS) like the one presented in this paper provide great flexibility and accessibility for process control engineers. Still this implies implementing specific measures for ensuring the safety, security and data integrity of such a connection.

The main aspects that need to be considered when designing such a system are [8]:

- Assuring the integrity of the data received by the remote device;
- Securing remote access by use of data encryption methods;
- Authenticating and validating the communication nodes.

The most common action [8 - 11] for solving these problems is using a dedicated VPN (Virtual Private Network) security solution. This solution implies the existence of a specialized VPN router at the plant level acting as a server that uses a public IP. At the router level the process control engineer will define a user and a password that will be used by the Library controller to authenticate as a VPN client.

While this solution minimizes the data loss and integrity, additional safety measures need to be considered when connecting to an industrial plant. For example, a network loss during an algorithm execution dynamic can lead to an instable process. Handling such cases may imply dedicated plant analysis, developing fail-safe mechanisms for the algorithms and implementing delay-handling algorithms as presented in [11]. Such mechanisms were not yet developed for this system and this is why it is intended for use only in low risk applications.

5. Use Case Example

To be able to execute an algorithm remotely using the presented web system it must first be adapted to a standard representation based on IEC 61499. We chose for exemplification the Particle Swarm Optimization (PSO) algorithm.

5.1 The PSO algorithm

Particle Swarm Optimization is a Swarm Intelligence (SI) method that solves optimization problems based on the analysis of the social behaviour of swarms like bird flocking or fish schooling [12]. It seems that these animals share information inside the swarm and the behaviour of each individual is dictated by patterns of the group behaviour in their search for food or in case of migration. The same way humans use both personal experience and other people's experience in a decision-making process. The PSO algorithm uses these concepts in optimization problems with D variables. Studies show that it is a fast, robust and easy to implement method able to find the global optima in continuous nonlinear optimization problems [12 - 14]. In process control applications it was often used for the online and offline tuning of PID control loops where studies shown increased performance compared to other tuning methods [14]. Enhanced hybrid versions of this algorithm can be used for solving problems of increased complexity like the Probabilistic Travelling Salesman Problem (PTSP) [15].

Let us consider we want to find the solution X^* for a minimization problem for a function f with D variables. The PSO algorithm uses a swarm of P particles that are placed with random positions x_i and random velocities v_i in a D-dimensional space. Each particle knows its own best position so far, *Pbest*, and the best

position of the group, *Gbest,* in the corresponding search space. The best position refers to the position for which the minimum of the function f was reached.

The velocity of each particle is modified according to its personal experience, the distance to the *Pbest* location, and also to the group experience, measured in the distance to the *Gbest* location. The modified velocity and position for each particle can be calculated using the following formulas (adapted from [13]):

$$v_{ij}^{t+1} = v_{ij}^{t} + c_1 * r_{1j}^{t} * [Pbest_{ij}^{t} - x_{ij}^{t}] + c_2 * r_{2j}^{t} * [Gbest_{j}^{t} - x_{ij}^{t}] \quad (1)$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t} \quad (2)$$

$i = 1, 2 \ldots P; j = 1, 2 \ldots D; t = 1, 2 \ldots N$

where:

N is the number of iterations of the algorithm;

v_{ij}^{t} and x_{ij}^{t} are the velocity and position vectors of particle i in dimension j at time t;

is the position vector of particle i in dimension j at time t;

$Pbest_{ij}^{t}$ is the personal best position of particle i in dimension j found from initialization through time t;

$Gbest_{j}^{t}$ is the global best position of all particles in dimension j found from initialization through time t;

c_1 and c_2 are acceleration constants;

r_{1j}^{t} and r_{2j}^{t} are random numbers generated by the algorithm between (0, 1) at time t.

The algorithm stops when the maximum number of iterations *N* is reached. The solution to the optimization problem is:

$$X^* = Gbest = (Gbest_1, \ldots Gbest_D) \quad (3)$$

A large swarm size *P* means an increased search space, which may also lead to a smaller number of iterations needed. Still, it will increase the computational complexity per iteration. Studies [9] have shown that a common value for *P* is inside the interval [20, 60]. The number of iterations also affects the algorithm efficiency. This is why a recommended practice is to define the maximum number of iterations *N* and to stop the execution of the algorithm either when that number is reached or when the algorithm stalls for a fixed period. The constants *c1* and *c2* express the self-confidence and the swarm confidence. These are often set to c1 = c2 = 2 according to past experiences [14].

5.2 Design of the PSO algorithm based on IEC 61499 function blocks

The choice of the IEC 61499 for the algorithm representation came from the need of creating a reusable algorithm, in an open standard, that can be used "as is" in different process control applications independent of the application development environment and of the platform on which it will run. The possibility of creating distributed applications further strengthened our choice since it gives the process control engineer the freedom to use advanced, complex algorithms using controllers with lower resources. This can be done by executing the complex algorithms remotely and by adding communication interfaces that link the controller in the field to the remote execution server [14].

Considering a minimization problem of the function f, the flowchart in Figure 2 illustrates the steps needed in the algorithm implementation, also according to the IEC 61499 execution.

INIT and REQ are input events that launch the execution of the corresponding sequences. All variable in the *_ij format should be read for all * variables where i goes from 0 to P, the number of particles, and j goes from 0 to D, the number of variables or dimensions of the problem. Dmin_j and Dmax_j is the interval in which the search should take place for each dimension. t keeps track of the current iteration. RUNF is a variable needed to separate the execution of the algorithm from the function evaluation.

This allows writing a generic, reusable algorithm that does not depend on the objective function for which it is used. Fbest represents a vector keeping the minimum values of the objective function at moment t and Pbest has the corresponding positions for which those values were obtained. Fgbest stores the global optimum and Gbest has the j values corresponding to the positions for which that optimum was reached. v_ij and x_ij are evaluated according to the equations (1) and (2). All other variables have the meaning specified in the previous section. The algorithm stops when the maximum number of iterations was

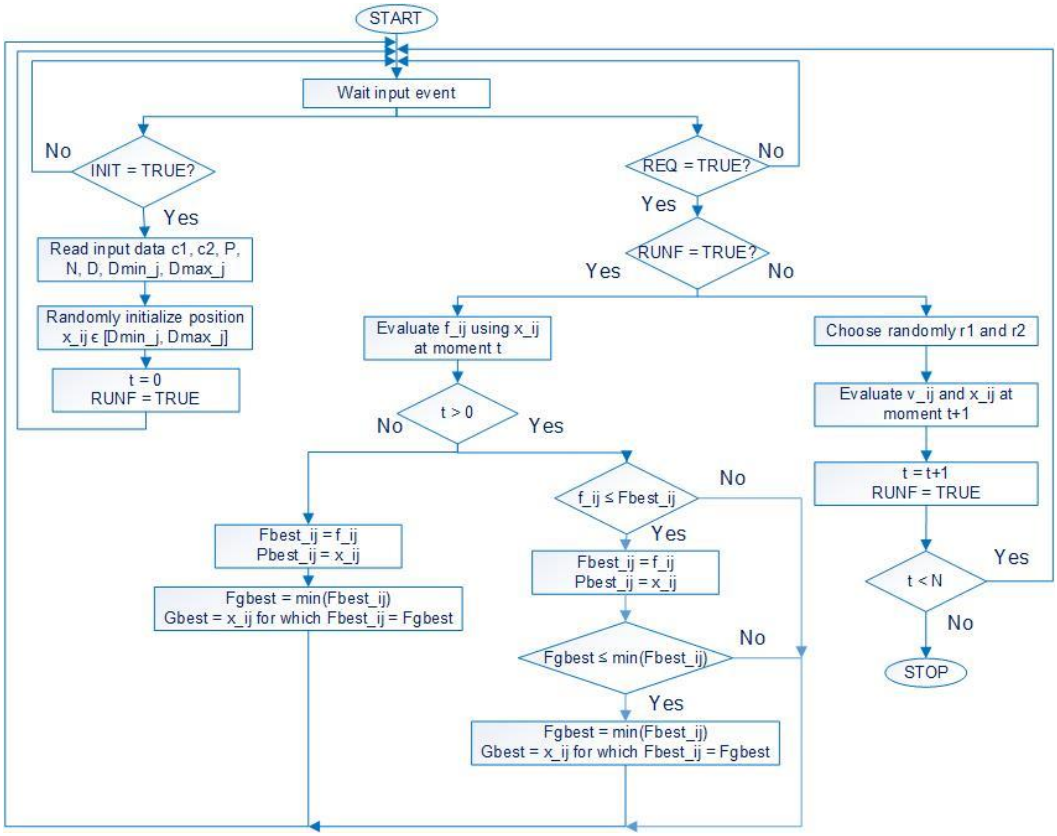

Figure 2. Flowchart of the PSO algorithm

reached. The outputs of the algorithm are the Gbest and Fgbest variables.

5.2.1 Algorithm implementation

The algorithm was implemented as a function block with the structure illustrated in Figure 3. The algorithm can be used for one dimension optimization problems. Because FBDK does not allow defining arrays with variable length, the maximum recommended length of 60 was considered for all arrays. This will not affect the execution performance as the algorithm will be evaluated according to the maximum number of particles P defined as input.

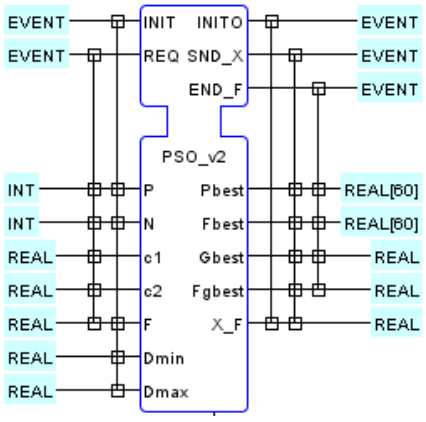

Figure 3. Algorithm structure

X_F is the value of X sent for evaluation to the objective function. F is the value received from the objective function. k is an internal variable that keeps track the current item in the position vector X to send to function f. All other variables have the same meaning as detailed in the above section.

The INIT event is used to launch the INIT function that initializes the algorithm variables and then sends an INITO output event.

When receiving an REQ event, the algorithm checks to see the value of the RUNF variable. If it is false it will execute the STEP function which is a step of the algorithm, meaning the evaluation of the velocity and position vectors. If the maximum number of iterations is reached, variable DONE is set to TRUE which activate the END_F output event.

The evaluation of f(X_F) is received at the input of the function block as variable F and is compared to Fbest[k]. If the new value is smaller, than X_F and F are stored in Pbest[k] and Fbest[k], respectively. When k reaches the end of the position vector (k=P) the RUNF is set to FALSE, allowing the function block to execute a new step of the algorithm. After a STEP function is executed, RUNF is set to

TRUE, the t variable is incremented and the k variable is reset. When the maximum number of iterations is reached the END_F output event is generated and the algorithm stalls until a new INIT event.

The algorithm can be used for the optimization of functions with more than one variable, considering the variables are independent of each other. Such an application is illustrated in Figure 4. Because the PSO algorithm is generic, different instances of it can be used for each variable. In this case the operator chooses the swarm size, the number of iterations and the search space dimension that are used for all algorithm instances. The Cyclic_sel block counts from 0 to the total number of needed input events and commands the Select block which activates cyclically each algorithm. The function f is executed according to the input event that is activated. For example, by activating the REQ 1 event, the block executes the function f(x1) and sends the result to z1.

network configuration illustrated in Figure 5 of three computers, each representing the different system components. The user accessed the Web interface from a computer where also the Library Server resides. Two computers were also used for simulating different remote devices. Because the IEC 61499 standard is platform independent, we were able to simulate two plant applications on remote computers. The functionality of the remote device does not affect in any way the behavior of the system, as their only connection is unidirectional from the Algorithm Execution module to the plant. The quality of the communication is only affected by the network performance.

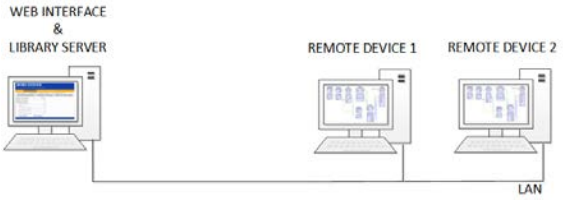

Figure 5. System evaluation configuration

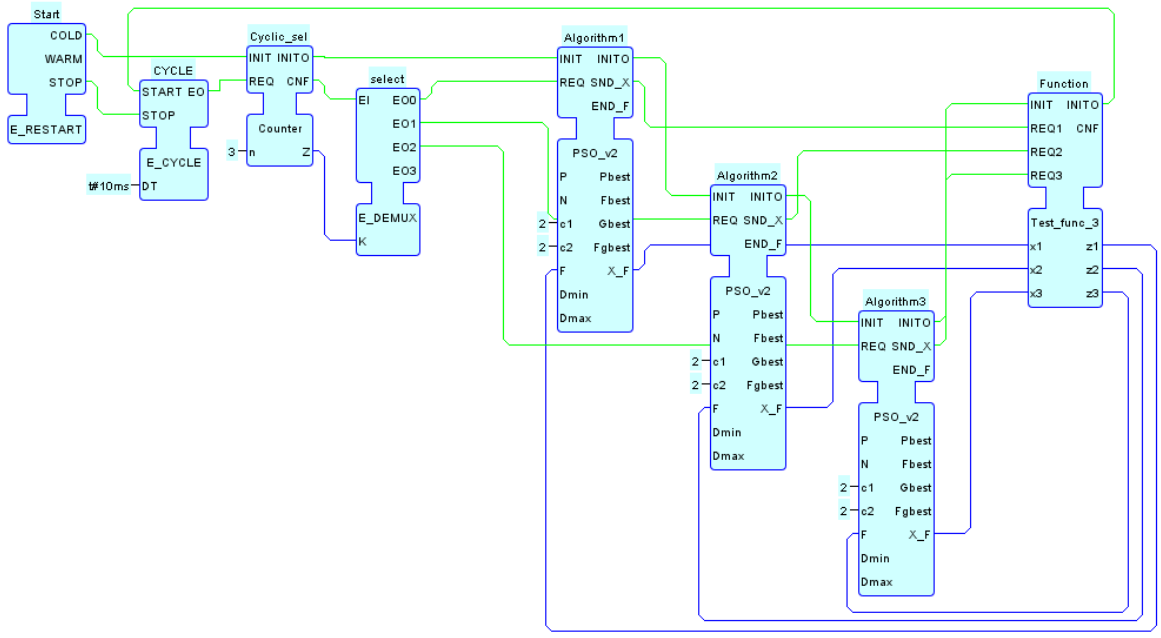

Figure 4. Optimization of a three variable function

6. System Testing and Evaluation

The web system for the remote control and execution of IEC 61499 applications was evaluated related to achieving the desired scalability and to how the system load in term of concurrent connections and/or concurrent algorithm executions affect the overall performance. For this evaluation we used the

To be able make this test our test harness had to spawn multiple algorithm instances and multiple plant modules. The communication between the algorithm and the plant is established though publish-subscribe blocks and each instance of the algorithm must have the publisher block communicate with the subscriber block through a unique IP/TCP Port tuple. The test harness is using the PSO

algorithm as the Algorithm Execution module and a system configuration simulating a simple process for the plant module. Both the algorithm and the plant module are represented in FBDK as .sys files which are handled by the System Manager library.

To have the same communication tuple between the Algorithm Execution module and the plant module, for these tests the Algorithm Execution module and the plant module are edited in the FDBK Editor and instead of assigning an IP/TCP port like 225.0.0.1:1024 for the publisher/subscriber blocks between the Algorithm Execution and the plant modules, we have assigned a pattern string that will be replaced by the Bibliodas Controller at runtime with a generated IP:Port tuple.

Once the pattern strings are replaced with the IP:Port tuples in both the plant module system file (.sys) and the Algorithm Execution system file, the Bibliodas Controller will first start the plant module by calling the SystemManager library with the specified system file. After a two seconds delay, time in which the FBDK runtime libraries will be able to open the listening sockets for the subscriber blocks and the plant module to bootstrap its configuration, the Bibliodas Controller will start the Algorithm Execution module.

Once the Algorithm Execution module is running, the Bibliodas Controller needs to send the algorithm parameters that the user has entered in the Web Interface to the Algorithm Execution module. The algorithm parameters are sent over a custom made publisher block from inside the Blibliodas Controller to the subscriber block in the Algorithm Execution module.

In our tests, we have monitored the CPU and Memory usage when simulating 1, 10, 20, 30, 40 and 50 concurrent algorithm sessions of the same algorithm. Each command is having the same algorithm parameters entered in the Web page and a unique session id, simulating different user sessions. The delay between each command was set to one second, so that the Bilibodas Controller will not be flooded with commands.

Test results are summarized in Figure 6. In all of the cases, the CPU usage was of about 30% and the Memory consumption was increased by 20MB for 10 simulated sessions and of maximum 40MB when simulating 50 concurrent sessions. The system responsiveness was slow at some point during the test, when multiple windows were opened and the taskbar in Windows XP was resizing to accommodate the new windows.

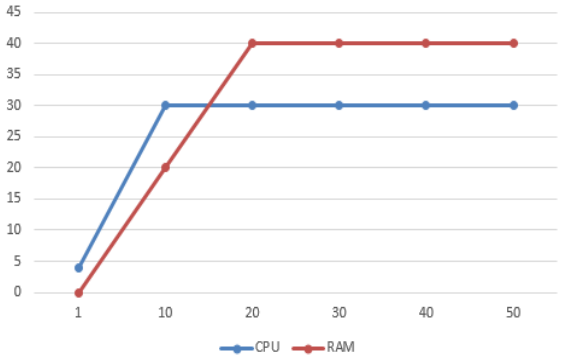

Figure 6. Performance test results

7. Conclusions

This paper presents the design and implementation of a web-based system for the remote control and execution of IEC 61499 function blocks. The system represents a new area of application for the IEC 61499 standard and uses different open technologies that allow system portability and easy components integration. The modular approach allows easy adaptability to different algorithms and applications while keeping the generic and easy reconfigurable characteristics. The paper details a possible solution for implementing such a system that uses different tools and programming languages specific for each component so that a flawless integration in achieved. The system extends the programming capabilities of IEC 61499 and gives process control engineers a starting point in designing other web-based or remote execution applications like cloud-computing for high complexity plants, increased availability using web-based HMI, integration of power system into Smart GRID networks etc. As the FBDK application used does not require great resources, the system scalability showed little influence on the server CPU and Memory load.

Future work is related to implementing system and communication security mechanisms, developing more advanced communication protocols that will allow integrating process feedback in the web interface and designing a more complex application that will be tested on a real application.

REFERENCES

1. SOROURI, M., S. PATIL, V. VYATKIN, **Distributed Control Patterns for Intelligent Mechatronic Systems**, International Conference on Industrial Informatics - INDIN, 2012.
2. ANDREN, F., T. STRASSER, A. ZOITL, I. HEGNY, **A Reconfigurable Communication Gateway for Distributed Embedded Control Systems**, Proc. of 38th Annual Conference of the IEEE Industrial Electronics Society 2012, October 25-28, 2012, Montreal, Canada.
3. LEWIS, R. W., **Modelling control systems using IEC 61499: applying function blocks to distributed systems**, Control Engineering Series 59, IEE, U.K., 2001.
4. STRASSER, T., J. H. CHRISTENSEN, A. VALENTE, F. CHOUINARD, E. CARPANZANO, A. VALENTINI, H. MAYER, V. VYATKIN, A. ZOITL, **The IEC 61499 Function Block Standard: Launch and Takeoff**, ISA Automation Week 2012.
5. VYATKIN, V., **IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design**, ISA Publishing, USA, 2012.
6. YANG, C., V. VYATKIN, **Design and Validation of Distributed Control with Decentralized Intelligence in Process Industries: A Survey**, Industrial Informatics, 2008.
7. ANDREN, F., T. STRASSER, A. ZOITL, I. HEGNY, **A Reconfigurable Communication Gateway for Distributed Embedded Control Systems**, Proc. of 38th Annual Conference of the IEEE Industrial Electronics Society 2012, October 25-28, 2012, Montreal, Canada.
8. JUNIPER Networks, **Architecture for Secure Scada and Distributed Control System Networks**, White paper, 2010.
9. JADHAV, M., G. GIDVEER, **Internet based Remote Monitoring and Control System**, International Journal of Advances in Engineering & Technology, March 2012.
10. KIRUBASHANKAR, R., K. KRISHNAMURTHY, J. INDRA, B. VIGNESH, **Design and Implementation of Web Based Remote Supervisory Control and Information System**, International Journal of Soft Computing and Engineering (IJSCE), Volume-1, Issue-4, September 2011.
11. TIPSUWAN, Y., M. Y. CHOW, **Control Methodologies in Networked Control Systems**, Control Engineering Practice, vol. 11, 2003.
12. KENNEDY, J., R. EBERHART, **Particle Swarm Optimization**, Proc. of the IEEE International Conference on. Neural Networks, vol. IV, Perth, Australia, 1995, pp. 1942-1948.
13. BAY, Q., **Analysis of Particle Swarm Optimization Algorithm**, Computer and Information Science, Vol. 3, No.1, 2010.
14. AGGARWAL, V., M. MAO, U. M. O'REILLY, **A Self-Tuning Analog Proportional-Integral-Derivative (PID) Controller**, First NASA/ESA Conference on Adaptive Hardware and Systems, 2006.
15. CABRERA, G. G., D. S. RONCAGLIOLO, J. P. RIQUELME, C. CUBILLOS, R. SOTO, **A Hybrid Particle Swarm Optimization - Simulated Annealing Algorithm for the Probabilistic Travelling Salesman Problem**, Studies in Informatics and Control, ISSN 1220-1766, vol. 21 (1), 2012, pp. 49-58.