Optimization of a milk processing application using a service oriented architecture

Alina Itu

Abstract – **Herein we present a service-oriented architecture, composed from three layers, which optimizes a milk processing application. The main objectives of the research were: minimizing the runtime, automated implementation of the optimized processing plans, and flexibility in the architectural composition. To minimize the runtime, a Mixed-Integer Programming (MIP) model is defined, and then solved with a Constraint Satisfaction Problem (CSP) solver. We also present in detail the structure of the address space defined for the OPC UA server, the controlling and the monitoring of the production through a complex service, and the Sequential Function Charts (SFC) run by the PLC. Various experiments are described in the results section, which indicate that the processing workload is evenly distributed to the workstations, and that the global runtime is minimized. Finally, we note that the computational overhead of the service-oriented architecture represents less than 1% of the global runtime.**

I. Introduction

The industrial environment has changed significantly over the last years, increasing the time-to-market demands. At the same time, numerous new technologies have been introduced, each one with its advantages and disadvantages, further enhancing the challenges faced by industrial players. Hence, to simplify the development of new solutions, processing and production approaches have to be time-oriented and –driven [1].

Consequently, the main requirements of manufacturing plants are nowadays [2]: dynamic integration – both intra- and inter-companies, agility – flexible and reconfigurable systems, and scalability – enhancing the capacity without having to halt the manufacturing process.

There are significant challenges for device networking ecosystems, that have to be addressed [3], [4]: adoption of web standards, secure, interoperable and universal device accessibility, reusable devices at any level, ensuring that each subsystem is exposed as a device which can be integrated in more complex systems, and each device must be manageable through a high-level interface, thus, simplifying monitoring, configuration, maintenance and fault diagnosis, as well as a predictable inter-device interaction.

Herein we make the case of the process optimization for a milk processing application, by employing a service oriented architecture composed from three layers, which was designed by taking into consideration the requirements described above. The objective was the development of a generic high-level description of the milk processing system, which leads to a minimized execution time, is reusable and flexible, and also performs alarm management and addresses maintenance aspects.

* This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CCCDI – UEFISCDI, project number ERANET-FLAG -RoboCom++ (2), within PNCDI III.

A. Itu is with the Transilvania University of Brasov, Department of Automation and Information Technology, Brasov, Romania (+40761642609, e-mail: alina.itu@unitbv.ro) and with Siemens SRL, Corporate Technology, Brasov, Romania.

II. Methods

A. Industrial Service Oriented Architecture

In the beginning a short introduction of the service oriented architecture previously described in [5] is provided. Specifically the architecture allows for the determination and automated usage of milk processing schedules that are optimized. The architecture displays the usual characteristics of service oriented architectures [6], i.e. autonomy and interoperability. Its overall concept is displayed in figure 1, where three main layers can be identified.

The bottom layer is represented by OLE for Process Control Unified Architecture (OPC UA) servers. These collect data from the control device, as well as from actuators and sensors. With the introduction of the UA specification of

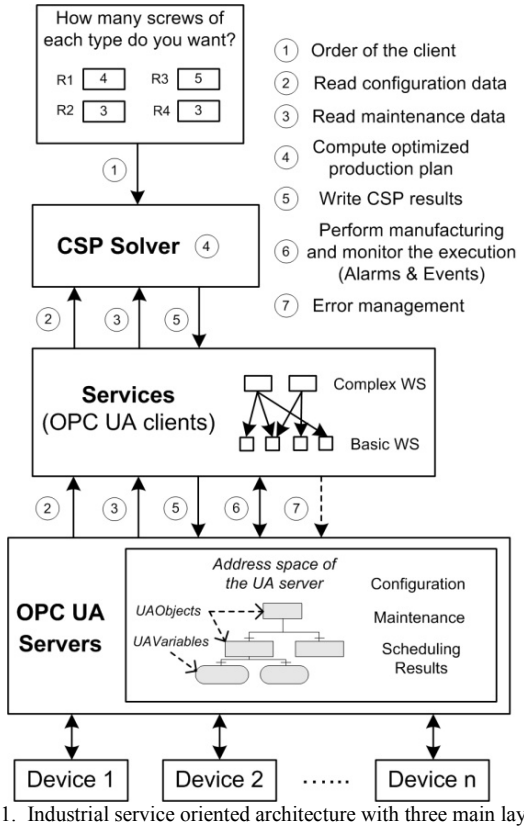

Fig. 1. Industrial service oriented architecture with three main layers.

OPC, industrial information can be modelled in standardized way, and real-time exchange of industrial data has become possible. The middle layer relies on software services of multiple types, which promote flexibility and adaptability. At the top layer, solvers based on the paradigm of Constraint Satisfaction Problems (CSP) determine the optimized production schedules and plans.

Moreover, the interaction between the three layers is also exposed in figure 1:

- *Step 1*: an order is received from a client, or is placed at the Enterprise Resource Planning (ERP) level;
- *Steps 2, 3:* the current state of the manufacturing devices is determined (software services communicate with the UA server);
- *Step 4*: an optimized manufacturing plan, taking into account the current state of the devices, is determined;
- *Step 5*: a complex service is started, which writes the optimized solution in the address space of the UA server;
- *Step 6*: the complex service monitors the execution of the optimized plans*;*
- *Step 7*: if an alarm is generated / an error occurs, it is managed by the complex service.

An important preliminary step is the automated generation of the UA server address space, by employing five previously introduced algorithms [7]. Besides the complex services mentioned above, the architecture also relies on basic services, performing more fine grained operations with the UA server, e.g. read or write. Complex services are also employed to consolidate Key Performance Indicators (KPIs) addressing the manufacturing process. At the CSP level, the most relevant concepts are the solver (allowing for the determination of an optimized manufacturing solution), and the model (generic representation of the manufacturing process).

B. Optimization of the milk processing application

1) Problem formulation

Three different milk types can be produced in a milk processing plant (fat content of 3.5%, 2.5% and 1.5%), by employing up to three workstations. The core components of each workstation are an inverter and an electrical asynchronous motor. The milk type is determined by the motor velocity. Specifically, we address the configuration displayed in table I: the production capacities for each type of milk and for each station are given. All types of milk can be produced on station 1, whereas only two types of milk can be produced on stations 2 and 3. Given a certain input order, the overall objective is to produce the milk, according to the order, in a minimal amount of time. An MIP model is formulated, having as outputs the processing time of each station at each velocity.

TABLE I
MILK PROCESSING CAPACITY OF EACH STATION

Milk type	Station 1 [l/min]	Station 2 [l/min]	Station 3 [l/min]
1,5%	10	12	0
2,5%	8	6	10
3,5%	6	0	5

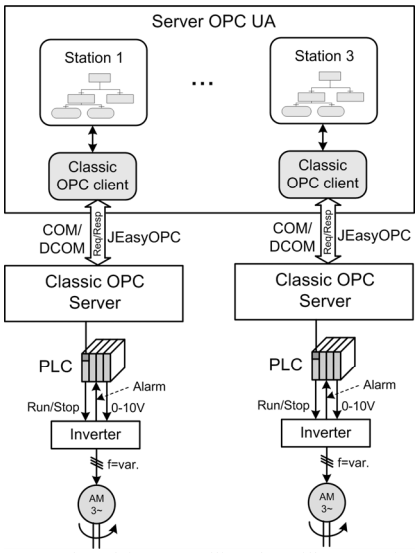

Fig. 2. Structure employed for controlling the milk processing application.

The control and test structure is depicted in figure 2 (the top and middle layers are omitted to simplify the representation). Specifically, we employed Siemens Micromaster 420 inverters and Bonfiglioli Riduttori eletrical motors, with a power of 0.37kW. Furthermore, the application has been abstracted in the address space of the UA server. The connection between the PLCs and the UA server relies on the special adapter software solution [8]: the OPC UA server embeds a classic OPC UA client to enable the communication with the control devices.

The inverter is controlled and monitored by two binary signals (*run-stop* command and *alarm* signal) and an analogue signal (4-20mA), which sets the reference velocity of the electric motor.

2) CSP model

Linear programming is a popular subtype of CSP [9], defined by the fact that only linear constraints are formulated. Specifically, a Mixed Integer Programming (MIP) model is formulated, relying on integer and binary variables [10]. The model may be adapted directly for any processing configuration, being thus generic.

The milk quantities defined in the order are used to initialize constants in the array *order*[*n*], where *n* is the number of milk types that can be produced. An additional two-dimensional array contains the constants representing the production capacities: *cap*[*m*][*n*], where *m* is the number of stations, representing another dynamic quantity similar to *n*. As mentioned above, the CSP solver needs to determine for each velocity and for each station the processing times. Hence, another two dimensional array is defined: *t*[*m*][*n*], where any value between zero and infinite may be set.

In a dynamic environment, stations may not always be available, e.g. due to maintenance works or errors. The station availability is stored in a one dimensional binary array, *avail*[*m*].

Three constraints are formulated to enforce that the overall milk quantities of each type, processed on the available workstations, correspond to the order:

$$order[i] = \sum_{j=1}^{m} avail[j] \cdot cap[j][i] \cdot cap[j][i]50; i = 1..n \quad (1)$$

where *j* refers to the stations, and *i* to the type of milk.

The overall processing time on each station will be stored in the array *t_total*[*m*]. To guarantee that the addition of the processing times specified in the array *t*[*m*][*n*], for a certain station, is the same as the processing time saved in the array *t_total*[*m*], a set of constraints needs to be defined, as follows:

$$t\_total[i] = \sum_{j=1}^{n} t[i][j]50; i = 1..m \quad (2)$$

This approach ensures that, even if different milk specifications are considered for a station, the results will be correct (given that the processing of different types of milk on a station is performed in sequence).

To determine a single solution of the MIP model, an objective function needs to defined. Hence, we introduce a variable, named *t_max*. As objective function of the MIP model we set the minimization of the largest processing time on the individual stations. Thus, we introduce the constraints:

$$t\_t\_total \le t\_\max; \quad i = 1..m \quad (3)$$

Consequently, the model minimizes *t_max*. Concretely, the JLPI (Java Linear Programming Interface) library, currently under development, was employed for model solving. This library allows for a dynamic choice between three MIP solvers, which are open-source: SoPlex, SCIP and GLPK.

3) Details of the implementation

For the milk processing application the address space depicted in figure 3 has been developed. A set of algorithms for automatically generating UA address spaces has been previously described [7], which has been reused herein. The entry point of the address space is represented by the objects: *Root*, *Object*, *Devices*. The information related to the processing stations is organized under the object called *Stations.* The start and end times for a certain manufacturing process are represented using the variables *Start* and *End*, linked directed to the *Devices* object.

Three objects associated with the different types of milk to be generated are defined for each *Station* object (fat content of 1.5%, 2.5% or 3.5%). The motor velocity which ensures the generation of the different types of milk is stored under these objects, alongside the nominal capacity of each station for each type of milk (values are in l/min and reflect the specifications in table I). The nominal capacity is set to 0 if that type of milk cannot be produced by that station.

For PLC compatibility, the velocities of the motor are directly stored in converter units (a Siemens S7-300 PLC was mounted in this case, having analogue outputs of type voltage, between 0 and 10V, which control the inverter). Identical structures have been chosen for the different stations, which allows for independent modifications to be made, and this leads to an improved flexibility during

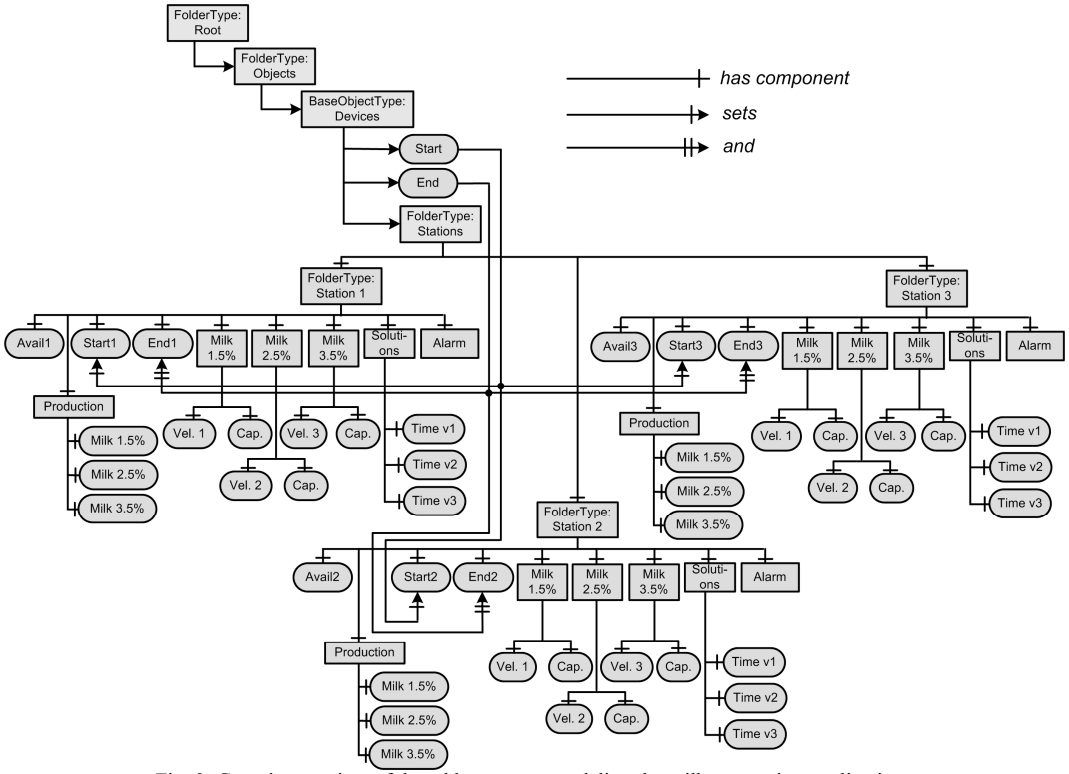

Fig. 3. Generic overview of the address space modeling the milk processing application.

manufacturing.

A *Solution* object is defined for each station, which stores the optimal solution computed by the MIP solver: it contains the runtimes for each motor velocity. These runtimes are then employed by the PLC during the actual processing of the milk.

Furthermore, a start variable (called *Start i*), and an end variable (called *End i*) have been defined for each station. When the overall *Start* variable becomes true, the station specific *Start i* variables are also set to true. These variables remain in the *true* state only for approximately 100ms, time after which they are reset to false. This ensures that the manufacturing process does not start over after having finalized the current instance.

Conversely, the *End i* variables are set to true when the milk processing has been finalized on station *i*. Once all *End i* variable become true, also the overall *End* variable becomes true. In detail: each variable or object is associated to a main reference, but may also have several other secondary references. In this case, the *set* references which connect the overall *Start* variable and the station specific *Start i* variables are secondary references. This type of reference (hierarchical reference) is defined in the UA specification and was introduced specifically for this application: it ensures that variables placed hierarchically lower have an identical value to the variables placed hierarchically higher. Another secondary reference which is not defined in the UA specification, and was introduced specifically for this application is called *and*. It links the overall *End* variable and the station specific *End i* variables and ensures that the variable placed hierarchically higher is equal to the logical *and* operation of the values associated to the variables placed hierarchically lower (it can only be employed for boolean variables). Thus, the overall *End* variable becomes true only if all *End i* variables are true, meaning that the processing was finished on all stations. Each *Station i* object also contains a boolean variable called *Avail i*: it defines the availability of the processing station (any station may not be available at a certain time point, either due to planned maintenance work or due to an error). Moreover, a *Production* object has been introduced for each station object, which stores the overall amount of milk which has been processed (it is used for performance monitoring at the ERP level).

Finally, each station also contains an *Alarm* object, set to true if an error has been signalled by the inverter.

4) Complex service

Apriori to running the MIP model, the processing times and the availability of the stations are determined through the read variable node service (a basic software service). Next, the MIP model is initialized based on these values, and based on the details of the order placed by the client. Subsequently, the MIP model optimization is run, and the optimal manufacturing plan is determined. Afterwards, the complex service displayed in figure 4 is started, which performs the functions of controlling and monitoring the milk processing. To fulfil its task, the complex service connects to the UA server, and sets up subscriptions for the *End* variable, and for the alarm objects.

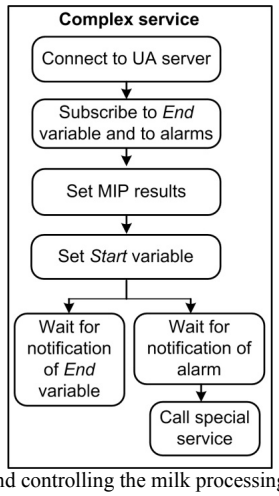

Fig. 4. Monitoring and controlling the milk processing application using a complex service.

The complex services receives the MIP results (in this case the values in the array $t[m][n]$), and writes them to the variable nodes of the workstations (for each station: *Time v1*, *Time v2*, *Time v3*). Next, the milk processing is started by setting the *Start* variable to true. Since the service had previously subscribed to the alarms, it can stop the execution if something unforeseen happens, case in which it will call a special service. For a normal execution, when the processing of the milk is finished, the *End* variable becomes true, effectively stopping the complex service execution.

5) The Control Graph of the PLC

The conceptualized SFC (Sequential Function Chart) run by the PLC for a station *i* is displayed in figure 5. When stage one (initial stage) is active, the inverter receives a stop command, and the *End i* variable is set to true, to indicate that milk is currently not being processed on the workstation. Afterwards, once *Start* is set to true in the address space, i.e. *Start i* becomes true, the motor velocity is set to v_{1i} (equal to the *Vel. 1* UA variable in the address space)), meaning that milk with a 1.5% fat content is being generated. Also, the *Run* signal, effectively starting the inverter, is set to true. The processing time t_{1i}, which defines the execution runtime of the motor at the above specified speed, is identical to the value of the *Time v1* variable of the corresponding station. The processing proceeds with the 2.5% and 3.5% fat content milk (of course, if one of the processing times is zero, the SFC moves immediately to the next processing stage).

The runtimes are measured in minutes and the conditions of transitions two, three and four are of the type *t/y/z*, where *t*

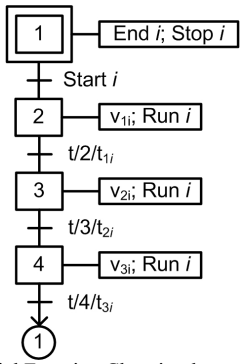

Fig. 5. Sequential Function Chart implemented in the PLC.

indicates that the transition is passed once a pre-specified time interval has elapsed after the activation of stage *y* (the exact interval of time is represented by *z*).

III. Results

The MIP solutions, computed for various orders, are listed in tables II, III, IV and V. We have chosen one order with identical quantities of milk, and three orders for which one type of milk has a larger quantity.

The processing times on the three workstations are similar for all four orders, indicating that the MIP solutions distribute evenly the workload on the different workstations, thus minimizing the overall execution time. There are four main modules in the application, and the processing times for each module and for each order are displayed in table VI. The vast majority of the processing time corresponds to the actual milk processing step (more than 99% of the time).

Modules one and three correspond to the services level in the architecture, module two corresponds to the CSP level, and module four to the UA server, service and device level.

TABLE II
PROCESSING QUANTITIES AND PROCESSING TIMES DETERMINED FOR THE THREE WORKSTATIONS WHEN CONSIDERING THE FOLLOWING ORDER: 1000 L MILK 1.5%, 200 L MILK 2.5%, 200 L MILK 3.5%

Station	**Time 1.5% [min] (litres [l])**	**Time 2.5% [min] (litres [l])**	**Time 3.5% [min] (litres [l])**	**Total time per station [min]**
Station 1	40.0 (400)	0.0 (0)	10.0 (60)	50.0
Station 2	50.0 (600)	0.0 (0)	-	50.0
Station 3	-	20.0 (200)	28.0 (140)	48.0
			Total execution time [min]	50.0

TABLE III
PROCESSING QUANTITIES AND PROCESSING TIMES DETERMINED FOR THE THREE WORKSTATIONS WHEN CONSIDERING THE FOLLOWING ORDER: 200 L MILK 1.5%, 1000 L MILK 2.5%, 200 L MILK 3.5%

Station	**Time 1.5% [min] (litres [l])**	**Time 2.5% [min] (litres [l])**	**Time 3.5% [min] (litres [l])**	**Total time per station [min]**
Station 1	2.0 (20)	26.0 (208)	30.0 (180)	58.0
Station 2	15.0 (180)	42.0 (252)	-	57.0
Station 3	-	54.0 (540)	4.0 (20)	58.0
			Total execution time [min]	58.0

TABLE IV
PROCESSING QUANTITIES AND PROCESSING TIMES DETERMINED FOR THE THREE WORKSTATIONS WHEN CONSIDERING THE FOLLOWING ORDER: 200 L MILK 1.5%, 200 L MILK 2.5%, 1000 L MILK 3.5%

Station	**Time 1.5% [min] (litres [l])**	**Time 2.5% [min] (litres [l])**	**Time 3.5% [min] (litres [l])**	**Total time per station [min]**
Station 1	2.0 (20)	1.0 (8)	90.0 (540)	93.0
Station 2	15.0 (180)	32.0 (192)	-	47.0
Station 3	-	0.0	92.0 (460)	92.0
			Total execution time [min]	93.0

TABLE V
PROCESSING QUANTITIES AND PROCESSING TIMES DETERMINED FOR THE THREE WORKSTATIONS WHEN CONSIDERING THE FOLLOWING ORDER: 700 L MILK 1.5%, 700 L MILK 2.5%, 700 L MILK 3.5%

Station	**Time 1.5% [min] (litres [l])**	**Time 2.5% [min] (litres [l])**	**Time 3.5% [min] (litres [l])**	**Total time per station [min]**
Station 1	2.0 (20)	1.0 (8)	90.0 (540)	93.0
Station 2	15.0 (180)	32.0 (192)	-	47.0
Station 3	-	0.0	92.0 (460)	92.0
			Total execution time [min]	93.0

TABLE VI
RUNTIMES OBTAINED FOR THE DIFFERENT ARCHITECTURAL LEVELS, CORRESPONDING TO THE ORDERS IN TABLES II, III AND IV.

Nr.	Module	Order 1		Order 2		Order 3		Order 4	
		Time [s]	Perc. [%]	Time [s]	Perc. [%]	Time [s]	Perc. [%]	Time [s]	Perc. [%]
1	Read the configuration data	2.61	0.09	2.65	0.07	2.64	0.05	2.60	0.05
2	Compute the MIP solution	0.32	0.01	0.33	0.01	0.32	0.01	0.34	0.01
3	Write the MIP solution in the address space of OPC UA server	2.32	0.08	2.33	0.07	2.31	0.04	2.34	0.04
4	Actual processing	3000.3	99.8	3480.4	99.8	5400.2	99.9	5580.3	99.9

IV. Conclusions

Herein we have described the optimization of an application for milk processing, by employing a three layered service oriented architecture. The processing application is modelled generically, and leads to a minimized processing time. The evaluation performed based on four concrete orders has shown that the workload is correctly distributed, and the computational overhead corresponding to the solving of the CSP model is minimal (the actual processing still represents around 99% of the overall runtime). Past researches have shown that a trade-off between performance and model is required during the CSP model definition. If the CSP model is very specific for the given task, the runtime may be short, but, if changes are required, the maintenance time will increase correspondingly.

On the other hand, in case of more generic models, the model solution runtime increases, but the flexibility of the model is enhanced. Herein, we have chosen to focus on designing a rather generic model, given that the runtime required to compute the optimized solution is anyway orders of magnitude faster than the actual manufacturing time. As a result, the different types of milk, as well as the number of stations, can be modified without requiring changes in the MIP model.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CCCDI – UEFISCDI, project number ERANET-FLAG - RoboCom++ (2), within PNCDI III.

References

[1] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska, "Experiences with GRIA – industrial applications on a web services grid," 1st Inter. Conf on e-Science and Grid Computing, Melbourne, July 2005, pp. 98-105.

[2] F. Jammes, and H. Smit, "Service oriented paradigms in industrial automation", IEEE Trans. on Industrial Informatics, vol. 1, pp. 62-70, Feb 2005.

[3] H. Bohn, A. Bobek, and F. Golatowski, "SIRENA - service infrastructure for real-time embedded networked devices: a service oriented framework for different domains," Proc. of the Inter. Conf. on Networking, Inter. Conf. on Systems, and Inter. Conf. on Mobile Communications and Learning Technologies, April 2006, pp. 43-47.

[4] A. Sasa, M.B. Juric, and M. Krisper, "Service-oriented framework for human task support and automation," IEEE Trans. on Industrial Informatics, vol. 4, pp. 292-302, Nov. 2008.

[5] A. Girbea, C. Suciu, S. Nechifor, and F. Sisak, "Design and implementation of a service-oriented architecture for the optimization of industrial applications", IEEE Trans. on Industrial Informatics, vol. 10, pp. 185-196, Feb. 2014.

[6] T. Cucinotta, A. Mancina, G.F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," IEEE Trans. on Industrial Informatics, vol. 5, pp. 257-266, Aug. 2009.

[7] A. Girbea, S. Nechifor, F. Sisak, and L. Perniu, "Efficient address space generation for an OPC UA server", Software-Practice and Experience, vol. 42, pp. 543-557, 2012.

[8] A. Girbea, S. Nechifor, F. Sisak, and L. Perniu, "Design and implementation of an OLE for process control unified architecture aggregating server for a group of flexible manufacturing systems", IET Software, vol. 5, no. 4, pp. 406-414, Aug. 2011.

[9] T. Sawik, Scheduling in supply chains using mixed integer programming. Hoboken, NY: Wiley, 2011.

[10] Y. Chu, F. You, "Integration of scheduling and control with online closed-loop implementation: Fast computational strategy and large-scale global optimization algorithm", Computers & Chemical Engineering, vol. 47, no. 20, pp. 248-268, 2012.