**Preprints of the 19th World Congress**
**The International Federation of Automatic Control**
**Cape Town, South Africa. August 24-29, 2014**

# A Model-based Approach for Achieving Available Automation Systems

**R. Priego* A. Armentia* D. Orive* E. Estévez** M. Marcos***

**University of the Basque Country, Department of System Engineering and Automatics, Bilbao, 48013 Spain (e-mail: rpriego001, aintzane.armentia , dario.orive, marga.marcos@ehu.es)*
*** Universidad de Jaén, Dept. Of Electronics and Automatic Control , Jaen, Spain (e-mail: eestevez@ujaen.es)*

**Abstract:** Nowadays automation systems are required to be flexible in order to cope with the ever changing requirements of the applications in terms of complexity, extensibility or dynamism. The use of reconfiguration techniques helps to meet these demanding requirements, but at the same time increase the complexity of the systems. This paper presents a supervisor architecture that allows maintaining the availability of a control system in spite of failures such as a node crash, by means of reconfigurations at the control level. To do this, the advantages that Model-Driven Development technology provides have been explored.

## 1. INTRODUCTION

Current manufacturing systems need to assure high productivity and a rapid response to fast changes in the market and customers needs. To meet these requirements the industry is evolving into more flexible and efficient manufacturing systems, for example providing them with reconfiguration mechanism. Hence, at it is stated in (Koren *et al.*, 1999) the introduction of reconfigurations allows changing as quick and cost-effectively as possible from one configuration to another, without stopping. Additionally, reconfiguration also helps to keep system efficiency from sudden changes of costumer demands or unpredictable events, like failures or disruptions. Even more, the National Research Council identified reconfigurable manufacturing as one of the six key challenges in manufacturing for the year 2020 (CVMC *et al.*, 1998).

Many works have been developed to provide manufacturing systems with the necessary reconfiguration techniques. Most implement new hardware equipments, such as modern soft Programmable Logic Controller (PLC), or try to incorporate technologies developed in other engineering fields, such as software engineering, model-driven development or distributed systems, to cover this necessity. Others have been focused in the reconfiguration of the whole plant like Fu-Shiung Hsieh (Hsieh, 2010) that implements a holonic manufacturing system to deal with resource failures and maintain order in the system.

However, in the last years there has been a tendency to implement Multi Agent Systems (MAS) (Weiß, 1999) in the manufacturing system. MAS have been used for agent-based control (Leitão, 2009), for concurrent design (Shen *et al.*, 2004), for collaborative process planning (Zhang and Xie, 2006), and for agent-based distributed manufacturing process planning and scheduling (Weiming Shen *et al.*, 2006; Ouelhadj and Petrovic, 2008). The use of MAS in reconfiguration can also be found in the lower levels of the control hierarchy. In this context, reconfiguration is based on function blocks (FB) of IEC 61499 standard (IEC, 2004), by changing the connections between FBs, changing their functionality, or adding and removing them (Brennan *et al.*, 2002; Lepuschitz *et al.*, 2011; Xu *et al.*, 2002).

Since manufacturing systems are getting more and more complex, increasing in terms of distribution, size, and functionality, the use of model-driven development (Selic, 2003) has been adopted to guide the user in the construction of the control system. For example, the use of Unified Modelling Language (UML) to describe the domain concepts is quite spread, as in the works focused on the IEC 61131 standard (Estevez *et al.*, 2005; Hästbacka *et al.*, 2011), or on the IEC 61499 standard (Thramboulidis *et al.*, 2006; Vyatkin and Hanisch, 2009). Another possibility is the use of SysML, an extended subset of UML 2, but applied to automation systems (Thramboulidis, 2011, Schütz *et al.*, 2013). However, these works do not deal with the reconfiguration needs of the automation systems.

This paper proposes the use of reconfiguration mechanisms together with model driven development techniques to tackle the issue of assuring the availability of control systems, in spite of PLC failures. Nowadays, the control functionality usually runs in a PLC, while the backup is another PLC that is waiting for the main to fail. This implies that the backup PLC underworks, as it is not being used until the main PLC fails. That is why there is a recent tendency to provide PLCs that not only run their own but also backup functionalities (Merz *et al.*, 2012). In this line the paper presents a supervisor and its corresponding design methodology that

assures the availability of the control system by restoring the functionality of a failed PLC into an already running PLC.

The layout of the paper is as follows: Section 2 describes the diagnosis and recovery processes proposed to maintain the availability of the control system. Section 3 details a model-based approach that guides the user during the design phase. Section 4 presents the supervisor architecture that assures availability in a control system which is used in Section 5 to illustrate a case study related to an assembly cell for rotation mechanisms. Finally, Section 6 is dedicated to the concluding remarks and future work.

## 2. AVAILABILITY OF THE CONTROL SYSTEM

System availability is assured when liveliness of the control system is maintained. To achieve this goal, the system must be provided with the necessary mechanisms to recover the functionality of a failed controller into another one, known as backup controller. Indeed, this section proposes a diagnosis and recovery processes that assure the availability of the control system based on the existence of backup controllers that, apart from their own functionality, also contain the functionality related to another controller. These "extra functionalities" are not executed in normal conditions only in case of failures.

A concrete point in the execution of a functionality is defined by the value of a set of relevant variables that represent the *state* of the functionality. A state can be composed by input and output variables, as well as internal variables like system variables, timers, counters, or variables representing GRAFCET (GRAphe Functionel de Commande Etape Transition) steps. When a fail is detected, a recovery process is started, i.e. the functionality of the failed PLC is restored in one of its backup PLCs. Three different ways to restore the functionality are distinguished, direct recovery, recovery to checkpoint, and non-recovery. (see Table 1). The type of recovery is determined by a previous diagnosis process.

The diagnosis is based on the last known state of the failed functionality, identifying the corresponding recovery type. As a result, all the information necessary to perform the recovery is obtained:

- Direct recovery: only the last known state is necessary, as nothing else needs to be done.
- Recovery to checkpoint: the state that describes a previously defined checkpoint from where the functionality will be restarted. A recovery action that prepares the system to be recovered from this checkpoint could also be necessary.
- Non-recovery: the necessary stop action that guides the functionality to stop in a safely manner.

The recovery process selects one of the available backups for executing the functionality and the information obtained during the diagnosis is sent to it.

**Table 1 Recovery types**

| Recovery type | Definition |
|---|---|
| Direct recovery | The control functionality can be resumed directly launching it in a backup controller from the last known state. |
| Recovery to checkpoint | The control functionality is resumed in a previous known state (checkpoint) after executing recovery actions if needed. |
| Non-recovery | It may require a safe stop procedure. |

## 3. DESIGN PHASE

As the information needed to perform the diagnosis and recovery processes is generated during the design phase, this section illustrates a model-based design methodology that guarantees the development of a proper control system in terms of availability.

The different functionalities that conform a control system are designed and generated following the Methodology for Industrial Automation (MeiA) (Alvarez *et al.*, 2013). As a result of applying this methodology the design of the execution of a plant is divided in a set of Design Organization Units (DOUs). These DOUs are the base for coding the Programmable Organization Units (POUs) which contain the code of a PLC functionality according to the IEC-61131-3 standard (IEC, 2003). This design is not only referred to the execution of the plant in normal conditions, but it also takes into account the stop and recovery actions needed if a fail occurs, i.e. the critical execution points.

The DOUs defined by the MeiA methodology are implemented as POU and deployed into different automation projects (one for each PLC in the system). These automation projects are generated following the XML schema provided by Technical Committee 6 of the PLCopen organization (Marcos *et al.*, 2009; Von der Wal, 2009). However, every PLC can also be backup of others. This means that its automation project has to be extended with the backup functionalities and the corresponding stop and recovery actions.

In summary, as it is shown in Fig. 1, during the design phase all the information related to the plant automation design is defined, identifying not only the DOUs but also the information about critical execution points. Additionally, the structure of the automation projects related to the PLCs that conform the control system is provided. Lastly, it is necessary to determine the backup functionalities that correspond to every PLC. All this information is transformed into the final automation projects that will run in the PLCs, and the necessary diagnosis information for assuring system control availability.

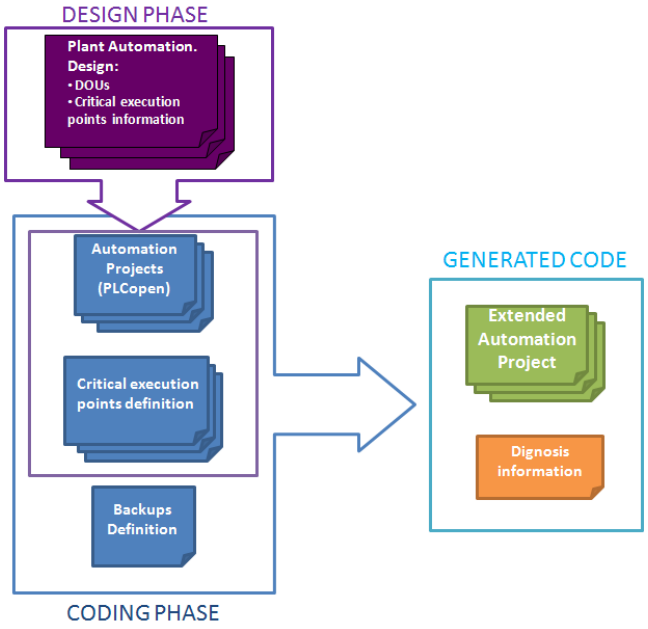


Fig. 1 Transformation of information

Fig. 2 depicts the meta-model related to the information coming from the design phase in Fig. 1. As it has been previously stated, the design of the execution of a plant is implemented in a set of *DOUs* and it has to take into account the different *critical execution points*. A *critical execution point* identifies a specific point in the execution that can not be directly recovered in case of failure. Therefore, they may be recovered to a checkpoint or even non-recoverable.

A *critical execution point* can be related to one or more *DOUs*, and a *DOU* can have one or more *critical execution points*. An execution point is defined by a set of *internal signals*, *I/O signals*, or *steps* belonging to several DOUs. Each *signal* is characterised by its *name*, *type*, and *value*.

When the *execution point* can be *recovered to a checkpoint*, it may be necessary to determine the *DOU* that performs the corresponding recovery action. Additionally, it is necessary to set the value of the *internal signals*, *I/O signals*, or *steps* which correspond to the checkpoint that is going to be restored. In the same way, *non-recoverable execution points* are provided with the *DOU* which stops the failed part or the whole plant in a safe way.

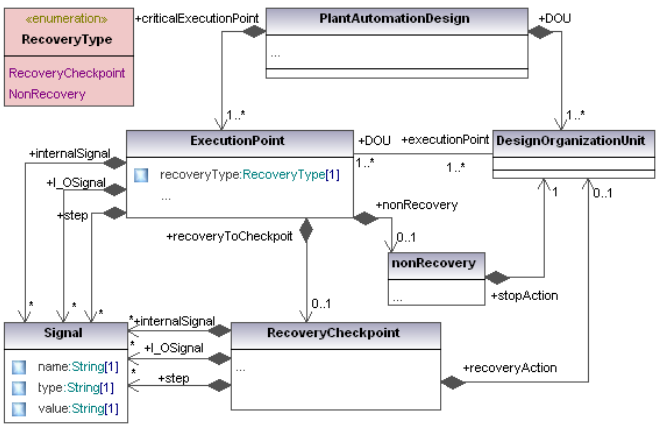


Fig. 2 Meta-model of the plant automation design

It is important to remark that the diagnosis information related to each PLC functionality is composed by the critical execution points that belong to all the DOUs within the functionality.

## 4. THE SUPERVISOR ARCHITECTURE

Traditional supervisor systems are based on Supervisory Control And Data Acquisition (SCADA), which inform the user about the state of the process and alarm event. The supervisor architecture presented in this section is a high level supervision that assures system availability, implementing the diagnosis and recovery processes previously described.

The supervisor architecture has been presented in (Priego *et al.*, 2013) using components running under the Distributed Component Management Platform (DCMP). The DCMP is an extensible middleware that manages distributed components (Agirre *et al.*, 2012).

The architecture is distributed into a supervisor PC which contains the Manager of the application (part of the DCMP), and several soft PLCs. It has been decided to use this PLC type as they can run concurrently operating system applications and PLC applications, meaning that the supervisor component can run in the same machine as the PLC functionalities (see Fig. 3). Additionally, since the workload of the manager is one of the heaviest ones, in terms of computational power, running it in a PC reduces the execution time.

The reconfiguration is decided by the manager, following the diagnosis and recovery processes previously described. But, the supervisor components are those in charge of performing

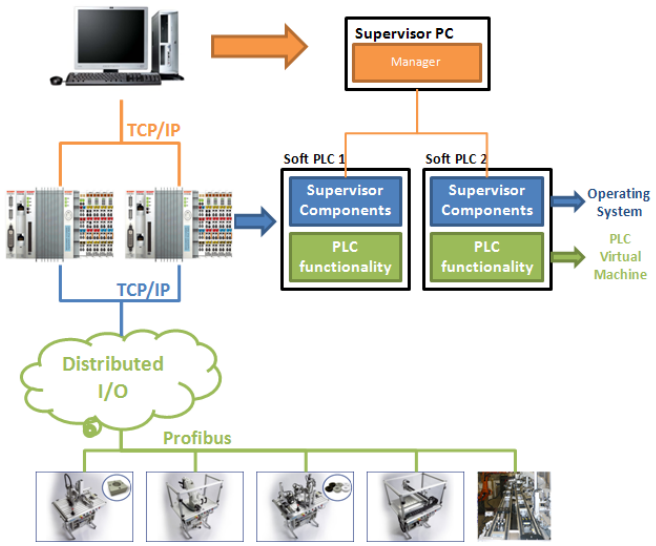


Fig. 3 General scenario of supervisor architecture

it. This requires to know if the PLC is alive and the state of the different functionalities.

The supervisor components link the PLC functionalities to the manager, sending the state that will be used during the diagnosis and recovery processes in case of failures.

Alongside the state these components also send a heartbeat as a liveliness proof. Indeed, the lack of the heartbeat allows the manager to detect that a PLC has failed. After, the manager proceeds to perform the diagnosis and recovery processes. The manager uses the last known state of the failed functionality determining the correct recovery type and the associated recovery information (as explained in previous sections). This information is sent to the corresponding backup. The supervisor components restore the functionality by writing the value of the state received into the corresponding variables, and starting the functionality depending on the type of recovery:

- Direct recovery: the functionality is restarted from the last known state.
- Recovery to checkpoint: recovery action is performed. After, the functionality is resumed from the checkpoint state.
- Non-recovery: stop action is performed which conducts the functionality to a safe stop.

It is interesting to note that the DCMP also provides a safe communication channel (Agirre *et al.*, 2013) that guarantees that the information has been received correctly. This safety channel is used to communicate the manager and the supervisor components.

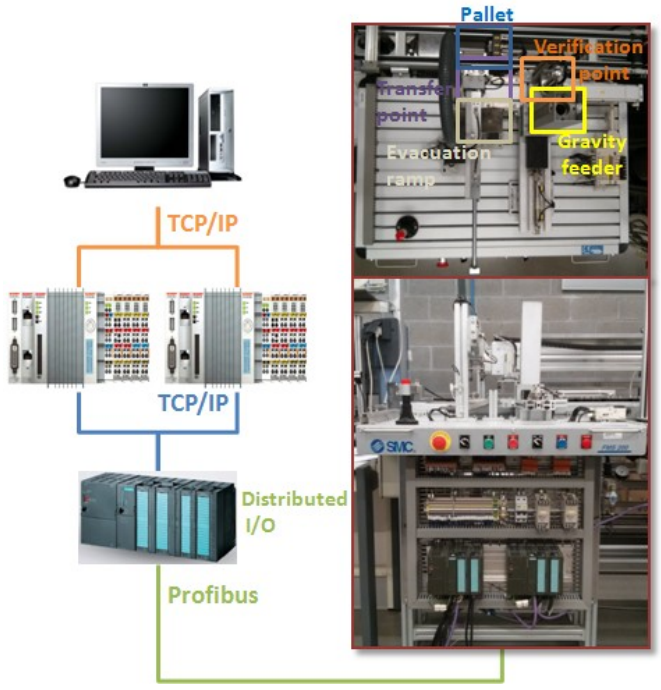


Fig. 4 Station 1: scenario

## 5. CASE STUDY: CONTROL AVAILABILITY FOR A MANUFACTURING STATION

The supervisor architecture has been tested in a station of an assembly line located in the Department of System Engineering and Automatics, in the University of the Basque Country. The assembly line produces different types of rotation mechanisms, composed of: a base, a bearing, an axis, and a cover. There are different types of, bearing, axis, and covers allowing the assembly of different types of mechanisms. The line is composed of four different stations: (1) the checking of the base, (2) the placement of the bearing and axis, (3) the selection of the cover, and (4) the storage of the final product.

The architecture is tested in the first station which is a FMS-201 from SMC (see Fig. 5). This station extracts a base from a gravity feeder moving it into the verification point, were the correct positioning of the base is checked. Then, the base is moved to the transfer point. If the base is incorrectly positioned it is removed from the station using the evacuation ramp. In the other case, it is moved to the transport pallet using vacuum pads.

To maintain the availability, the supervisor architecture is implemented by a PC and two soft PLCs (Beckhoff CX1020). As the I/O of the station are connected using Profibus, a Siemens PLC has been used to allow the change of the master during run time. The communication between the Beckhoff PLC and the Siemens PLC is done using

```
<?xml version="1.0" encoding="UTF-8"?>
<PlantExevutionDesign>
    <criticalExecutionPoint recoveryType="NonRecovery">                    1
        <step name="Control1.Sequence1_1.X21" type="BOOL" value="1"/>
        <step name="Control1.Extract_Base1.X51" type="BOOL"  value="1"/>
        <nonRecovery>
            <stopAction id="StopAction1"/>
        </nonRecovery>
    </criticalExecutionPoint>
    <criticalExecutionPoint recoveryType="RecoveryCheckpoint">             2
        <step name="Control1.Sequence1_1.X23" type="BOOL" value="1"/>
        <step name="Control1.Insert_Base1.X73" type="BOOL"  value="1"/>
        <recoveryToCheckpoint>
            <recoveryAction id="RecoveryAction1"/>
            <internalSignal name="Visualizacion.mascara" type="BYTE"
                value="-1"/>
            <internalSignal name="Control1.L_INIT" type="BOOL" value="1"/>
            <internalSignal name="Control1.L_PCI" type="BOOL" value="1"/>
            <internalSignal name="Control1.S_SEQ_EXT2" type="BOOL"
                value="1"/>
            <step name="Control1.Sequence1_1.X21" type="BOOL" value="1"/>
            <step name="Control1.Locate_Palet1.X40" type="BOOL" value="1"/>
            <step name="Control1.Extract_Base1.X50" type="BOOL" value="1"/>
            <step name="Control1.Check_base1.X60" type="BOOL" value="1"/>
            <step name="Control1.Insert_Base1.X70" type="BOOL" value="1"/>
            <step name="Control1.Lack_Base1.X10" type="BOOL" value="1"/>
            <step name="Control1.Remove_Palet1.X80" type="BOOL" value="1"/>
        </recoveryToCheckpoint>
    </criticalExecutionPoint>
    <criticalExecutionPoint recoveryType="NonRecovery">                    3
        <step name="Control1.Sequence1_1.X23" type="BOOL" value="1"/>
        <step name="Control1.Insert_Base1.X74" type="BOOL"  value="1"/>
        <nonRecovery>
            <stopAction id="StopAction2"/>
        </nonRecovery>
    </criticalExecutionPoint>
    <criticalExecutionPoint recoveryType="NonRecovery">                    4
        <step name="Control1.Sequence1_1.X23" type="BOOL" value="1"/>
        <step name="Control1.Insert_Base1.X75" type="BOOL"  value="1"/>
        <nonRecovery>
            <stopAction id="StopAction3"/>
        </nonRecovery>
    </criticalExecutionPoint>
</PlantExevutionDesign>
```

Fig. 5 Station 1: plant automation design model

TCP/IP.

During the design phase of the station, four critical execution points have been identified: three non recovery execution points and one recovery to checkpoint. Fig. 4 presents the plant automation design model for the station. This model is defined in an eXtensible Markup Language (XML) document that follows an XML Schema which implements the meta-model illustrated in Fig. 2. The first non-recoverable execution point (number 1 in Fig. 4) is related to the extraction of the base from the gravity feeder. The extraction of the base is performed by a single action cylinder. If the cylinder loses the action signal during the fail of the controller, the cylinder goes to its initial condition, and the base that has been moved can be placed in a position which obstructs the output of another base. This is a non-recovery execution point since the execution cannot be recovered until an operator removes the fault base; therefore, the stop action consists on informing the operator.

The second non-recoverable execution point (number 3 in Fig. 4) is associated to the movement of the base from the transfer point to the pallet. During this movement if a fail occurs the base is released from the vacuum pads and can fall in any position of the trajectory. In this case the operator also needs to remove the base and restart the process. The stop action is similar to the previous one, but in this case the cylinder that moves the base needs to be returned to its initial condition automatically.

The last non-recoverable execution point (number 4 in Fig. 4) refers to the moment when the base is placed in the pallet. If a fail occurs, the base can fall into an incorrect position in the pallet. The operator also needs to clean the station so the recovery action is similar to the previous one, but it is also necessary to place the cylinder into its original position.

The recovery to checkpoint execution point (number 2 in Fig. 4) occurs during the listing of the base to be moved to the pallet. In case of a failure, the base falls into the transfer point. Operator intervention is not necessary as the base can be evacuated using the evacuation ramp. So, the corresponding recovery action moves the base into the evacuation ramp, cleaning the system. Then, the normal functionality is restored to the extraction of the base from the gravity feeder (the check point).

Finally, all the information about these critical execution points is transformed into the diagnosis data used in the diagnosis process (see . Fig. 1). As commented before, when a fail occurs, the manager maintains the operation of the station, by recovering the functionality of failed PLC into the corresponding backup PLC. It performs the diagnosis of the last known state, giving as a result the recovery type.

## 6. CONCLUSIONS AND FUTURE WORK

The architecture proposed in this paper maintains the availability of a control system, by restoring the functionality of a controller into a backup, when a failure occurs. This allows optimizing the resources of the system, since it is not necessary the use of redundant PLCs

It has been highlighted that the recovery of the functionality depends on the execution point in which it has failed. Therefore, a previous diagnosis process has been proposed which determines the suitable way to recover the functionality, based on its last known state.

Aimed at guarantying availability, the paper also describes a model-based design methodology that assures the correct development of a control system. The starting point is the design of the control system, based on the MeiA methodology, which also contains the recovery and stop actions corresponding to each execution point. The plant automation model is obtained, which just contains the critical execution point of the system and the information needed for the recovery of these points.

As proof of concept, the supervisor architecture has been validated in a station of an assembly line. This station presents four critical execution points with their corresponding stop, and recovery actions and checkpoint states.

Future work will be focused on improving the generation process of the final code, by implementing a tool for defining the backup PLC, and generating the diagnosis information as well as the extended automation projects. Therefore, it will take as inputs the information provided during the design phase, the structure of the initial automation projects related to the functionalities, the plant automation design (DOUs, and critical execution points). Furthermore, other kinds of reconfiguration are been research in order to support other Quality Of Services (QoS) requirements, such as load balancing and energy efficiency

## ACKNOWLEDGMENTS

This work was supported in part by the MCYT&FEDER under grant DPI- 2012-37806-C02-01, the Government of the Basque Country (GV/EJ) under grant BFI-2011-251 and Project IT719-13 and by UPV/EHU under grant UFI11/28.

## REFERENCES

Agirre, A., Perez, J., Priego, R., Marcos, M. and Estévez, E. (2013). SCA Extensions to Support Safety Critical Distributed Embedded Systems. In: *IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA),* Cagliari, Italy.

Agirre, A., Marcos, M. and Estévez, E., (2012). Distributed applications management platform based on Service Component Architecture. In: *IEEE 17th Conference on Emerging Technologies Factory Automation (ETFA)*, pp. 1–4, Krakow, Poland.

Alvarez, M. L., Estévez, E., Sarachaga, I., Burgos, A. and Marcos, M. (2013). A novel approach for supporting the development cycle of automation systems. In: *The International Journal of Advanced Manufacturing Technology*, **vol. 68**, pp.711–725.

Brennan, R.W., Fletcher, M. and Norrie, D.H. (2002). An agent-based approach to reconfiguration of real-time distributed control systems. In: *IEEE Transactions on Robotics and Automation*, **vol. 18**, pp.444–451.

Committee on Visionary Manufacturing Challenges, Commission on Engineering and Technical Systems, National Research Council (1998) *Visionary Manufacturing Challenges for 2020*. (National Academies Press) Washington, DC.

Estevez, E., Marcos, M., Gangoiti, U. and Orive, D. (2005). A Tool Integration Framework for Industrial Distributed Control Systems. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 8373–8378.

Hästbacka, D., Vepsäläinen, T. and Kuikka, S. (2011). Model-driven development of industrial process control applications. In: *Journal of Systems and Software*, 84(7), pp.1100–1113.

Hsieh, F.-S., (2010). Design of reconfiguration mechanism for holonic manufacturing systems based on formal models. In: *Engineering Applications of Artificial Intelligence*, **vol. 23**, pp.1187–1199.

International Electrotechnical Commission (2003). IEC International Standard IEC 1131-3 Programmable Controllers, Part 3: Programming Languages

International Electrotechnical Commission (2004). IEC International Standard IEC 61499 Part 1.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. andVan Brussel, H. (1999). Reconfigurable Manufacturing Systems. In: *CIRP Annals - Manufacturing Technology*, **vol. 48**, pp.527–540.

Leitão, P., (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. In: *Engineering Applications of Artificial Intelligence*, **vol. 22**, pp.979–991.

Lepuschitz, W., Zoitl, A., Vallée, M. and Merdan, M. (2011). Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **vol. 41**, pp.52–69

Marcos, M., Estevez, E., Perez, F. and Van Der Wal, E. (2009). XML exchange of control programs. In: *IEEE Industrial Electronics Magazine*, **vol. 3**, pp.32 –35.

Merz, M., Frank, T. and Vogel-Heuser, B., (2012). Dynamic redeployment of control software in distributed industrial automation systems during runtime. In: *2012 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 863–868.

Ouelhadj, D. and Petrovic, S., (2008). A survey of dynamic scheduling in manufacturing systems. In: *Journal of Scheduling*, **vol. 12**, pp.417–431.

Priego, R., Armentia, A., Orive, D. and Marcos, M. (2013) Supervision-based Reconfiguration of Industrial Control Systems. In *18th Conference on Emerging Technologies Factory Automation (ETFA), 2013 IEEE*. Cagliari, Italy.

Schütz, D., Obermeier, M., and Vogel-Heuser B. (2013) SysML-Based Approach for Automation Software Development – Explorative Usability Evaluation of the Provided Notation, In: *Design, User Experience, and Usability. Web, Mobile, and Product Design* (Springer Berlin Heidelberg), **vol. 8015**, pp. 568-574.

Selic B. (2003) The pragmatics of model-driven development, In: *IEEE Software*, **vol. 20**, pp. 19 – 25.

Shen, W., Norrie, D.H. and Barthes, J.-P., (2004). *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing* (C. Press, ed.)

Thramboulidis, K., (2011). Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation. In: *Journal of Software Engineering and Applications*, **vol. 4**, pp.217–226.

Thramboulidis, K., Perdikis, D. and Kantas, S. (2006). Model driven development of distributed control applications. in: *The International Journal of Advanced Manufacturing Technology*, **vol. 33**, pp.233–242

Vyatkin, V. and Hanisch, H.-M., (2009). Closed-Loop Modeling in Future Automation System Engineering and Validation. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **vol. 39**, pp.17–28.

Van der Wal, E. (2009). PLCopen. In: *IEEE Industrial Electronics Magazine*, **vol 3**, pp 25.

Weiming Shen, Lihui Wang and Qi Hao, (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, **vol. 36**, pp.563–577

Weiß, G., (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, (Mit Press).

Xu, Y, Brennan, R W., Zhang, X. And Norrie, DH. (2002). A Reconfigurable Concurrent Function Block Model and its Implementation in Real-Time Java. In: *Integrated Computer-Aided Engineering*, **vol. 9**, pp.263–279.

Zhang, W.J. and Xie, S.Q. (2006). Agent technology for collaborative process planning: a review. In: *The International Journal of Advanced Manufacturing Technology*, **vol. 32**, pp.315–325.