A novel approach for Flexible Automation Production Systems

Elisabet Estévez
Electronic and Automatic Engineering Department
EPS de Jaén, University of Jaén
Jaén, Spain
eetevez@ujaen.es

Federico Pérez, Darío Orive, Marga Marcos
Dept. of Automatic Control and System Engineering
ETSI Bilbao, Basque Country University (UPV/EHU)
Bilbao, Spain
{federico.perez, dario.orive, marga.marcos}@ehu.eus

***Abstract*— Nowadays, the constantly changing market demands make modern automation systems offer more strict flexibility and efficiency requirements as energy efficiency, performance optimization and tolerance to process or controllers faults… The introduction of reconfiguration mechanisms within production automation systems allow meeting some of these requirements but, at the same time, they lead more complex design and implementation of the automation system. This paper explores the advantages of Model Driven Engineering techniques to manage such complexity. The proposed approach is based on two well spread standards in the automation field: AutomationML and IEC 61131-3 which have a Markup Language notation (AML and PLCopen XML respectively).**

Keywords—AutomationML, Flexible Automation Production Systems, Model Driven Engineering

I. INTRODUCTION

Nowadays, public institutions of most developed countries are investing on industrial manufacturing in order to encourage innovation, job creation and economic growth. These initiatives pursue the integration, reuse, flexibility and optimization of manufacturing processes by implementing high-tech features based on the use of adaptive and smart equipments and systems [1]. In this context, current automation systems have begun to exhibit mechanisms for dynamic reconfiguration to assure non-functional requirements. This work uses the term of reconfiguration as the ability to relocate the different functionalities over the distributed control systems in order to optimize some type of quality of service, e.g. controller performance, battery consumption or service continuity despite controller or network failures.

This implies an increase of their complexity in terms of size, functionality and distribution, making, as a consequence, the design and development processes more complex. In this context, the use of Model-Driven Engineering (MDE) has been proved suitable to guide and help in the design, development and implementation phases of complex systems [2].

This paper presents a MDE based approach that generates the necessary mechanisms to assure certain quality of service to automation systems. The use of Model Driven techniques in the realm of industrial automation is not new [3]-[8]. In the recent years, there are works that use models to extend the system characterization (e.g. Program Organization Units (POUs), controller, machine...) in order to allow system reconfiguration. [9] characterizes the software view with information related to resource demands (amount of memory and number of bytes exchanged with other Function Blocks-FB) and this information is used to decide the system nodes in which FBs deployed. [10] introduces functional and non-functional requirements as constraints to the different views of the production automation system, starting from a sensor or an actuator up to the entire plant, and a tolerance model that traces whether the required reliability will be maintained, and if the required probability of a given quality will be reached. This information is used by an agent system to provide sensor failure tolerance, by replacing a measurement of sensor with a computed one. [11] presents an aspect based characterization of the non-functional requirements of the functional components for embedded control systems.

The previous works are very interesting and demonstrate the usefulness of using MDE for very different purposes. Most of them capture both, functional and non-functional requirements in order to generate the final code that assures such requirements. The goal of this paper goes further as it deals with an extensible model-based approach that is able to generate the mechanisms needed to assure certain automation system quality of service. It is built upon on well-spread and accepted standards in the automation realm which have Markup Language notation: AutomationML [12] and PLCopen XML [13].

The remainder of the paper is as follows: Section 2 proposes an AML based modeling approach for modeling automation systems. Section 3 and 4 are dedicated to the automatic code generation. Section 3 details the generation of Flexible Automation Projects, meanwhile, section 4 depicts the procedure followed for generating application dependent code. The system flexibility assessment is performed in Section 5 through a case study. Finally, Section 6 outlines the most important conclusions.

II. RECONFIGURABLE AUTOMATION PRODUCTION SYSTEM AML BASED DESIGN

A Reconfigurable Automation System consists of a set of: (1) distributed controllers (PLCs); (2) Mechatronic Components (MCs) which each one manages a part of the process; (3) a runtime platform that is responsible for

978-1-5386-0837-1/17/$31.00 ©2017 IEEE

guaranteeing the fulfillment of automation system quality of service at runtime. This work assumes the Flexible Automation Middleware (FAM) as runtime platform. This is a generic multi agent system (MAS) that can be customized to monitor a set of system quality of service [14]. Reconfiguration means to de-activate MCs in some controllers activating them in others, changing the structure of the automation system. Hence, MCs are characterized by operation states: (1) *non-critical* states are those in which the automation system knows the actual state of the process and thus, MCs can be activated in another controller having as initial state the last known state of the deactivated MC. (2) *Critical states* are those in which the automation system does not know exactly the current state of the MC. Critical states inhibit activation/de-activation of MCs in order to avoid unpredictable process behavior. For instance, after a controller failure it is necessary to analyze if all MCs currently in execution in the failed controller can be recovered in a previous known state (*checkpoint*) or, on the contrary, it is not possible to recover from the fail and the MC must be safely stopped and the operator warned (*non-recovery*). Note that FAM is also responsible for assuring that at run-time, every MC will only be active in one controller. To do this, FAM includes four types of agents. Two of them are part of the basic architecture able of managing different system qualities: (1) Middleware Manager (MM)- ,(2) Supervisor that comprises Monitor agents (QMs) and a Diagnosis and Decision (D&D) agent; and other two that depend on the automation application: (3) Controller Agent (CA) for registering the corresponding controller and associated resources in a repository; and (4) Mechatronic Component Agent (MCA) for managing the execution of the corresponding MC control logic.

Developers Design and develop automation systems as a set of MCs that use a set of I/Os, POUs and variables. MCs are replicated in a number of controllers. Furthermore, each MC is characterized by a set of critical intervals defined by a condition. For the design phase, this work adds to AutomationML mechanisms to allow developers modeling the MC properties. The followed the procedure is detailed in [15]. By itself, AML provides a PLCopen interface that provides access to the different elements (POUs and variables). A new hardware interface has been added to enable the definition of the controllers. Fig. 1 shows an example of the use of the editor that defines a mechatronic component named MC1.

III. Flexible Automation Project Generation

Current IEC 61131-3 standard execution environments do not allow dynamic code deployment, preventing true reconfiguration of the application. Because of that, the flexible automation projects must contain all MCs the controller may run. This explains why a reconfiguration means de-activation of an MC in a controller and its activation in another.

Other aspect to be considered is the execution management of MCs. This management is performed by MC agent (MCA) that requires read and write the current state of the MC it manages. This is performed via specific sections of the controller memory. Hence, a Flexible Automation Project contains not only the functionality of the MCs but also the control code that provides their flexibility. This is achieved through three POUs which are automatically generated: (1) a control program that manages the execution of the MC (*MCid_Control*); (2) a program that collects and serializes the value of the variables that compose the state of the MC (*MCid_Serialize*); and (3) a program that de-serializes and fixes the values to initialize the state of a MC when it changes from not active to active in a controller (*MCid_Deserialize*).

The automatic generator of Flexible Automation Projects gets information from the AML model and using Model to Model transformation techniques generates those POUs in a PLCopen XML format.

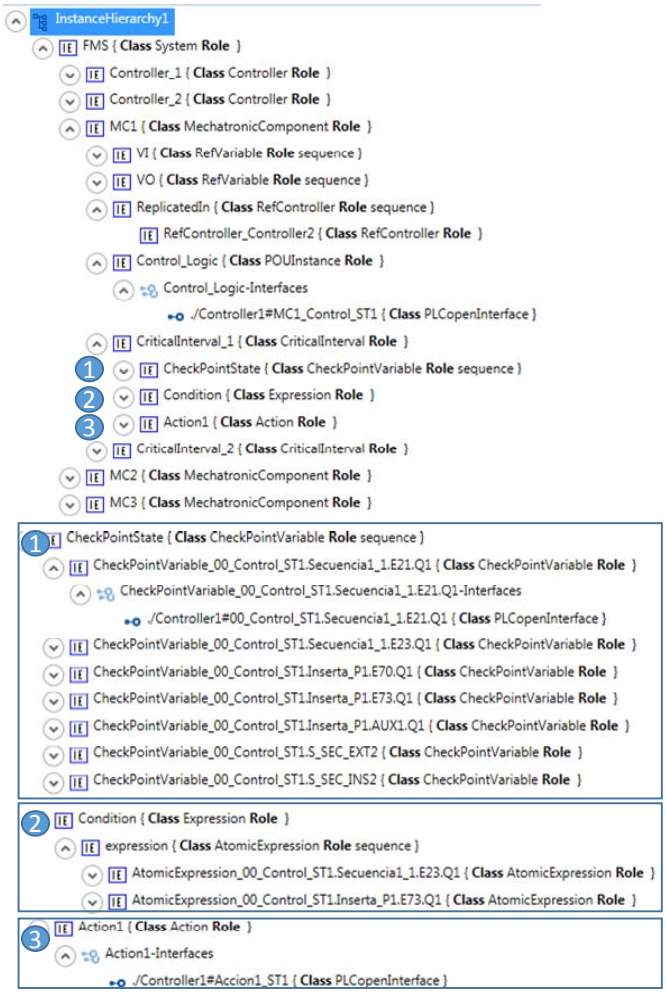

Fig. 1. Example of Automation control system design

The following sub-sections detail the followed procedure.

A. MC Execution Management

This is an automatically generated POU that allows the MCA to activate/deactivate the execution of the logic and to launch recovery/stop actions of the corresponding MC. It is composed by two main parts: interface and body.

The interface is a set of application dependent variables and other local static variables that allow MCA to manage it. These local variables are:

- *isActive* and *wasActive*: boolean variables for managing the activation/deactivation of the MC.

In order to support the availability, the following additional local variables are required:

- *recoveryAction*: it is related to the recovery code to execute for the critical interval if necessary..
- *Action_CriticalIntervalID*: It corresponds to the actual recovery code (POU instance).

On the other hand, the general structure of the body is the following:

```
IF isActive=TRUE and wasActive=TRUE THEN
    MCid(); /* MC enriched with the wrapper */
    MCid_Serialize();
ELSE IF isActive=TRUE and wasActive=FALSE THEN
    CASE recoveryAction OF /* the list value depends on the MC*/
      /* the list cases depends on the critical intervals of the MC */
    END_CASE
ELSE IF isActive=FALSE and wasActive=TRUE THEN
    wasActive=FALSE;
END_IF
```

Finally, other transformation rule completes the list of the possible cases when one MC is activated (isActive=TRUE and wasActive=FALSE). Table I illustrates the code template in ST to add in every case. Note that, the MCid is customized with the ID of the corresponding MC.

TABLE I. CODE TEMPLATES FOR NON-CRITICAL AND CRITICAL STATES

Direct recovery	Check Point Recovery	Safe Stop

MCid_Deserialize();
MCid ();
MCid_Serialize();
wasActive=TRUE;
	MCid_Action_id();
	IF(MCid_Action_id.end)
	THEN
	MCid_Deserialize();
	MCid ();
	MCid_Serialize();
	wasActive=TRUE;
	recoveryAction=0;
	END_IF
		MCid_Action_id();
		IF(MCid_Action_id.end)
		THEN
		recoveryAction=0;
		isActive=FALSE;
		END_IF

B. Serialization and de-serialization of the MC's state

The serialization program is able to gather the execution states into a byte array, which is accessible by the MCA.

The de-serialization program follows the same structure, but instead of collecting the information into an array, it extracts the information from an array provided by the MCA and rewrites the value of the state variables.

The interface of both POUs is the same. It is formed by a local variable which collects the data representing the state of the MC.

According with body, serialization program template in ST is the following:

```
state[0]:=TypeOfGlobalVariable_TO_BYTE(GlobalVariable);
…
state[NumberOfBytes]:=TypeOfGlobalVariable_TO_BYTE(GlobalV
ariable);
```

Finally, the body of de-serialized program is listed below:

```
GlobalVariableName=BYTE_TO_TypeOfGlobalVariable (state[0]);
…
GlobalVariableName = BYTE_TO_TypeOfGlobalVariable
(state[NumberOfBytes]);
```

IV. Generation of Application Dependent Agents

The procedure for generating application dependent agents is the following: first, two Java based templates have been defined, one for MCAs and other for CAs. Both java templates have a set of application dependent properties that are filled automatically getting the corresponding information from AML file. The following sub-sections details the generation of MCAs and CAs, as many MCAs per MC as controllers may run it and one CA per controller.

A. Mechatronic Component Agents (MCA)

As established in [14], MCAs implement a Finite Machine State. Each MCA is linked to a different MC residing in a PLC and it performs state diagnosis when required in order to determine if the current state belongs to a critical interval. Each state is associated to the corresponding functionality implemented as two JADE behaviors (SimpleBehaviour). Meanwhile, PLCAccessibility provides access to the MC code in the PLC.

It has two customizable parameters: *MC_ID*, which is the ID of the corresponding MC and state_diagnosis which addresses the set of masks that define the critical intervals of the MC. A relevant task in MCA generation is to transform critical interval information into a set of masks to diagnose the MC state. This information is generated and stored into the so-called Diagnosis XML file. This file contains the information of the variables that form the state (name and type), the masks for performing the diagnosis as well as the checkpoint states. The diagnosis masks filter the state variables related to the condition and allow determining the type of critical state (checkpoint or unrecoverable).

B. Controller Agents (CA)

The CA provides updated information about the state of the controller resources. They also participate in negotiation processes when needed. Negotiation criterion depends on the specific quality of service to be assured. The basic functionality of the CA is implemented in a cyclic behavior managing the messages from the middleware agents. On the other hand, the CA can implement resource monitoring behaviors that allow the monitoring a specific resource of the controller as part of the QoS monitoring process.

It has a set of customizable parameters: *ID* for identifying the agent in the system; a textual *Description*; a *CPUfactor* with respect to a reference controller; *memory* resources; and a list of MCs, which their control logic is executed in the controller (*AssignedMC*).

V. Case Study

The Flexible Automation projects modeling approach and automatic code generation proposed in this work have been applied to assure work balance among a set of distributed controllers in the automation system for the flexible assembly cell FMS-200. The assembly cell consists of four stations and a modular conveyor system (Transport Station) that is in charge of assembling a set of four pieces (base, bearing, shaft and lid). In the first station, the base that acts as the support is fed, its orientation is verified and, if correct, it is moved to the pallet located on the transfer system. In the second, the

bearing and shaft are placed on the base, while the cap is put in the third station. In the last station, the set mounted on the pallet is stored in the warehouse.

The cell is divided into five MCs, one for each station and other for the transport station. However, to simplify the assessment, only the MCs associated to the first three stations are going to be considered, MC1, MC2 and MC3 respectively. The demonstrator is comprised of two Beckhoff CX1020 controllers in which the MCs are executed. The demonstrator also contains a supervisor PC in which the Middleware Manager and the Supervisor will run.

The station has five critical intervals, with their corresponding actions that allow the station to continue extracting the lids from the warehouse. These MCs are modeled with the proposed AML based approach. Part of this model is illustrated in Fig.1.

After the code generation phase, the Flexible Automation Projects, CAs, MCAs and diagnosis information files are deployed into the corresponding controllers. Fig. 2 illustrates part of the Flexible Automation Project automatically generated for the controller 1 in PLCopen XML format.

Tests on reconfiguration have been performed by starting the control system in a single controller (controller 1) able to run the three MCs. During the execution, a new controller joins the system (controller 2). After registration, monitoring of the workload detects that it is under the specified limit. CA2 issues a work balance event to FAM which decides the new distribution based on the CPU factors of the CAs, current distribution of the MCs, as well as the maximum CPU workload introduced by the MCs. After obtaining the new distribution the relocation process begins.

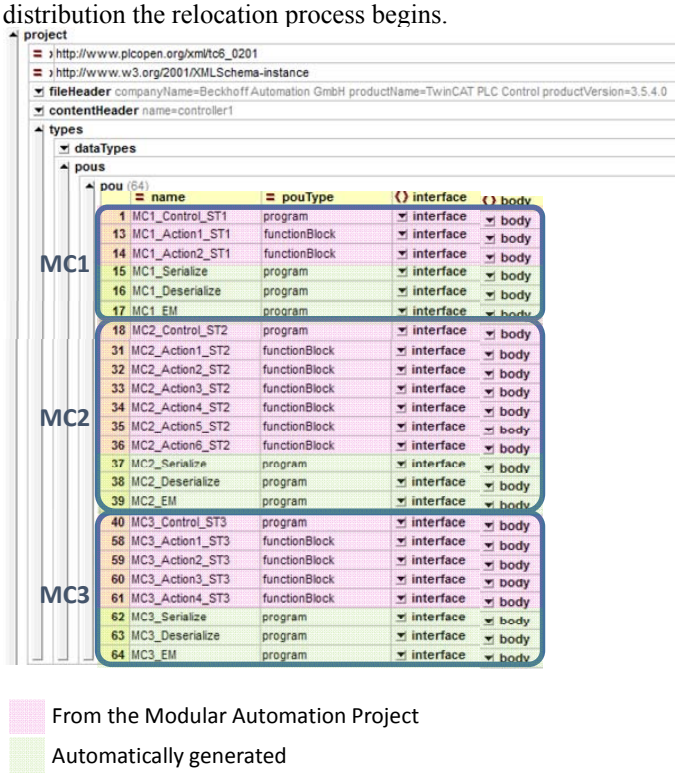

Fig. 2. Flexible Automation Project for controller 1

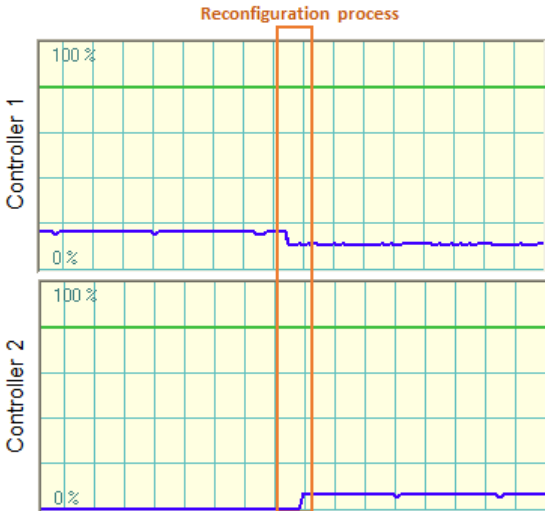

Fig. 3. Workload of controller 1 and controller 2 before and after the reconfiguration

During this process, the active MCA will stop the execution of the control code in a non-critical state. After the reconfiguration process, MC1 and MC2 are running in controller 1 and MC3 is running in controller 2. Fig. 3 presents a graphic depicting the workload of the different controllers before and after the introduction of controller 2.

VI. Conclusions

This paper presents an approach aiming at adding flexibility to automation system enabling thus control system reconfiguration. This work uses a MAS-based middleware that provides QoS management at run-time. Authors proposed a model based approach that captures relevant information about the production process and the distributed automation system as well as how to use it into valuable information for the management platform. The proposed approach adds flexibility to the original distributed automation system, supporting dynamic reconfiguration of the control system. In fact, the model-based approach presented in this work demonstrates the advantages of using models and model transformation not only to automate code generation but also to collect relevant information about the system that is fundamental in order to achieve dynamic reconfiguration.

Acknowledgment

This work was financed under project DPI2015-68602-R (MINECO/FEDER, UE) and GV/EJ under recognized research group IT914-16.

References

[1] Association EF of the FR. Factories of the Future PPP FoF 2020 Roadmap: Consultation document. 2012.

[2] Selic B. The pragmatics of model-driven development. IEEE Softw 2003;20:19–25. doi:10.1109/MS.2003.1231146.

[3] Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide (2nd Edition). Addison-Wesley Professional; 2015.

[4] Estevez E, Marcos M, Gangoiti U, Orive D. A Tool Integration Framework for Industrial Distributed Control Systems. Proc. 44th IEEE Conf. Decis. Control, IEEE; 2005, p. 8373–8. doi:10.1109/CDC.2005.1583518.

[5] Hästbacka D, Vepsäläinen T, Kuikka S. Model-driven development of industrial process control applications. J Syst Softw 2011;84:1100–13.

[6] Thramboulidis K, Perdikis D, Kantas S. Model driven development of distributed control applications. Int J Adv Manuf Technol 2006;33:233–42. doi:10.1007/s00170-006-0455-0.

[7] Vyatkin V, Hanisch H-M. Closed-Loop Modeling in Future Automation System Engineering and Validation. IEEE Trans Syst Man, Cybern Part C (Applications Rev 2009;39:17–28. doi:10.1109/TSMCC.2008.2005785.

[8] SysML. The SysML Specification, v 1.0 2007. http://www.sysml.org

[9] Fay A, Vogel-Heuser B, Frank T, Eckert K, Hadlich T, Diedrich C. Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. J Syst Softw 2015;101:221–35. doi:10.1016/j.jss.2014.12.028.

[10] Vogel-Heuser B, Rösch S. Integrated Modeling of Complex Production Automation Systems to Increase Dependability. Risk - A Multidiscip. Introd., 2014, p. 1–476. doi:10.1007/978-3-319-04486-6.

[11] Binotto APD, Wehrmeister MA, Kuijper A, Pereira CE. Sm@rtConfig: A context-aware runtime and tuning system using an aspect-oriented approach for data intensive engineering applications. Control Eng Pract 2013;21:204–17. doi:10.1016/j.conengprac.2012.10.001.

[12] AutomationML (2017) [website] http://www.automationml.org

[13] Marcos M, Estevez E, Perez F, Van der Wal E. XML exchange of control programs. IEEE Ind Electron Mag 2009;3:32–5. doi:10.1109/MIE.2009.934794.

[14] Priego R, Iriondo N, Gangoiti U, Marcos M. Agent Based Middleware Architecture for Reconfigurable Manufacturing Systems. Int J Adv Manuf Technol (2017). doi:10.1007/s00170-017-0154-z. In Press.

[15] Estevez E, Marcos M. Model-Based Validation of Industrial Control Systems. IEEE Trans Ind Informatics 2012;8:302–10. doi:10.1109/TII.2011.2174248.