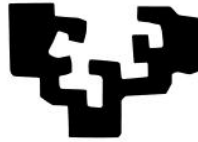


eman ta zabal zazu



Universidad del País Vasco      Euskal Herriko Unibertsitatea

**Department of Automatic Control and System Engineering  
UNIVERSITY OF THE BASQUE COUNTRY (UPV/EHU)**

**PhD Thesis**

# **A Model-based Approach for supporting Flexible Automation Production Systems and an Agent- based Implementation**

**AUTHOR**

**Rafael Priego Rementeria**

**Supervised by**

**Marga Marcos Muñoz  
Birgit Vogel-Heuser**

**This dissertation is submitted for the degree of Doctor of Philosophy  
Bilbao, Abril de 2017**



## ACKNOWLEDGMENT

---

I would like to express my gratitude to Prof. Marcos and Prof. Vogel-Heuser, which have provided guidance and help during the development of the PHD thesis. I will also would like to thank the people of the Department of Automatic Control and System Engineering who have participated in this work, especially Aintzane, Dario and Unai, as well as Eli from the university of Jaen.

I would not want to forget any of my colleagues, they have all had provided words and gestures of encouragement during this process, thank you all. A special thanks to Pablo, Aitziber, Sara, Ekaitz, Victor, Villa and Oier.

A special thanks to my parents, my brother and sister and my girl friend Bea that have suffered my ups and downs during these last years.

Thanks to all

This PHD thesis has finance by the Government of the Basque Country under the grand BFI-2011-251.

And has been developed as part of the projects:

INTEGRACION DE INTELIGENCIA DISTRIBUIDA Y SEMANTICA EN LA FACTORIA INTELIGENTE (MINECO REF **DPI2015-68602-R** (subprograma DPI)) Ministerio de Economía y Competitividad, FROM: 01/01/2016 TO: 31/12/2018

APLICACIONES DE FABRICACION RECONFIGURABLES DIRIGIDAS POR REQUISITOS DE QoS (MINECO REF **DPI2012-37806-C02-01** (subprograma DPI)) Ministerio de Economía y Competitividad, FROM: 01/01/2013 TO: 31/12/2015

Control Inteligente e Integración de Energías Renovables en Sistemas Eléctricos (CINIERSE). (Unidad de Formación e Investigación (UFI) REF **UFI1128**) Universidad del País Vasco (UPV/EHU), FROM: 1/01/2011 TO: 31/12/2014

SUBVENCIÓN GENERAL A GRUPOS DE INVESTIGACIÓN (REF **GIU07/36**) Universidad del País Vasco (UPV/EHU), FROM: 12/05/2008 TO: 11/05/2011

# ABSTRACT

---

Modern manufacturing systems are expected to be flexible and efficient in order to cope with challenging market demands. Thus, they must be flexible enough as to meet changing requirements such as changes in production, energy efficiency, performance optimization, fault tolerance to processes or controller faults, among others. Demanding requirements can be defined as a set of Quality of Service (QoS) requirements to be met.

This research work proposes a generic and customizable management architecture to deal with QoS loss during runtime. The QoS loss detection and reaction performed by the architecture can be customized for different QoS goals. As a proof of concept, the architecture has been implemented using a Multi Agent System middleware that makes use of distributed agents for monitoring QoS and triggering, if needed, a reconfiguration of the control system to recover the QoS.

However, the introduction of these mechanisms leads to more a complex designs and implementation of the control system. For this reason, this work tries to deal with this complexity by means of a model-based framework that helps and guides the definition and development of flexible automation systems. The framework provides a series of model-based tools that allow the automatic generation of control code extensions aiming at adding flexibility to the automation production system.

The prototype has been tested in a case study consisting of an assembly cell where assessment of the approach has been conducted.



## **INDEX**

---





---

# Content Index

<b>1 Introduction</b>	
1.1 Motivation .....	1-1
1.2 Research Goals .....	1-3
1.3 Structure .....	1-4
<b>2 Related work</b>	
2.1 Introduction .....	2-1
2.2 Manufacturing System Reconfiguration .....	2-3
2.3 Model Driven Engineering in Manufacturing System .....	2-10
2.4 Conclusions .....	2-13
<b>3 QoS managEmEnt architecture</b>	
3.1 Introduction .....	3-1
3.2 General Scenario.....	3-1
3.3 Production system information model .....	3-3
3.3.1 MC information model .....	3-3
3.3.2 Controller information model .....	3-8
3.3.3 Run-time platform model.....	3-12
3.4 Conclusions.....	3-15
<b>4 Agent based middleware Architecture</b>	
4.1 Introduction .....	4-1
4.2 Middleware Manager.....	4-4
4.3 QoS Supervisor .....	4-6
4.3.1 QoS Monitor (QM) Agents .....	4-6
4.3.2 Diagnosis & Decision (D&D) Agent.....	4-6
4.4 Application Agents.....	4-7
4.4.1 Controller Agent.....	4-7
4.4.2 Mechatronic Component Implementation .....	4-8
4.5 QoS Management Ontology .....	4-12
4.5.1 Registration messages .....	4-13
4.5.2 Information messages.....	4-15
4.5.3 QoS loss messages .....	4-17
4.5.4 Negotiation messages .....	4-18
4.5.5 Diagnosis messages.....	4-19
4.5.6 Reconfiguration messages .....	4-20

4.5.7 State message .....	4-21
4.6 Assuring QoS .....	4-21
4.6.1 Availability QoS.....	4-22
4.6.2 System Efficiency QoS.....	4-24
4.7 Conclusions.....	4-27
<b>5 The Flexible Automation Framework</b>	
5.1 Introduction .....	5-1
5.2 Generation of tool-independent automation projects .....	5-2
5.3 Flexible Automation Production System Model.....	5-5
5.4 AML-based System definition (FAPS Editor).....	5-6
5.5 Flexible Automation Projects .....	5-10
5.5.1 MCid_Control .....	5-11
5.5.2 Serialization and de-serialization of the MC's state.....	5-14
5.6 Application Agents.....	5-16
5.6.1 MC Agent Templates.....	5-17
5.6.2 Controller Agent Template .....	5-18
5.6.3 Diagnosis File.....	5-20
5.7 Conclusions.....	5-21
<b>6 Proof of Concept</b>	
6.1 Introduction .....	6-1
6.2 Manufacturing Demonstrator .....	6-1
6.3 Modular Automation System.....	6-4
6.4 Flexible Automation System Design.....	6-5
6.5 Run-time Performance.....	6-9
6.5.1 Availability QoS.....	6-9
6.5.2 System Efficiency QoS.....	6-12
6.6 Conclusions.....	6-13
<b>7 Conclusions and future works</b>	
7.1 Conclusions.....	7-1
7.2 Future works.....	7-4

**References**

**Glossary**

## Figure Index

Fig. 2-1 eCEDAC-Approach for Distributed Online Change (Schimmel & Zoitl, 2011) .....	2-9
Fig. 2-2 A Transfer FB moving an Agent FB from the origin device to the desired target device (Yan & Vyatkin, 2013) .....	2-10
Fig. 3-1 Flexible Automation Production System .....	3-2
Fig. 3-2 Example of critical interval .....	3-5
Fig. 3-3 Software View: concepts and their relationships .....	3-10
Fig. 3-4 Meta-model of the hardware concepts and their relationships .....	3-11
Fig. 3-5 Run-time platform .....	3-12
Fig. 3-6 QoS characterization meta-model .....	3-15
Fig. 4-1 Middleware Implementation .....	4-2
Fig. 4-2 Middleware Manager System Repository .....	4-4
Fig. 4-3 MCA Finite State Machine (FSM) .....	4-8
Fig. 4-4 Execution control code template .....	4-10
Fig. 4-5 Middleware Message Ontology .....	4-13
Fig. 4-6 Availability QoS Monitoring and QoS loss detection .....	4-22
Fig. 4-7 Availability loss detection .....	4-23
Fig. 4-8 Service recovery through negotiation phase .....	4-24
Fig. 4-9 System Efficiency QoS Monitoring and QoS loss detection .....	4-25
Fig. 4-10 System efficiency loss detection .....	4-26
Fig. 4-11 Relocation Service .....	4-26
Fig. 5-1 General Scenario of Flexible Automation Framework .....	5-1
Fig. 5-2 UML class diagrams and UML profiles for the different domain views .....	5-3
Fig. 5-3 General scenario of the UML modeling tool .....	5-5
Fig. 5-4 MC Meta-Model .....	5-6
Fig. 5-5 CAEX libraries for Flexible Automation Systems .....	5-8
Fig. 5-6 Flexible Automation control system design example .....	5-9
Fig. 5-7 Execution control program generation example .....	5-14
Fig. 5-8 Example of Flexible Automation Project .....	5-16

## Figure Index

---

Fig. 5-9 General Structure of System MCAs .....	5-17
Fig. 5-10 General Structure of System CAs.....	5-19
Fig. 5-11 General Structure of Diagnosis.xml files.....	5-20
Fig. 6-1 Manufacturing System Demonstrator.....	6-1
Fig. 6-2 Modular Automation Project Controller1 .....	6-4
Fig. 6-3 AML controller definition .....	6-5
Fig. 6-4 definition of MC1 in FAPS Model Editor .....	6-6
Fig. 6-5 Critical intervals and replication information of MC1 .....	6-7
Fig. 6-6 Flexible automation project for controller 1 .....	6-8
Fig. 6-7 Controller agent for Controller1 .....	6-8
Fig. 6-8 MC agent for MC1.....	6-8
Fig. 6-9 Diagnosis file for MC1.....	6-9
Fig. 6-10 Recovery Time vs. Number of Controllers Involved in the Negotiation .....	6-10
Fig. 6-11 Recovery Time vs. Number of MCs to be recovered .....	6-10
Fig. 6-12 Diagnosis time vs Number of critical interval masks .....	6-11
Fig. 6-13 Reconfiguration Time vs Number of MCs.....	6-13

---

## Table Index

Table 2-1 Control System Reconfiguration Approaches .....	2-14
Table 3.1 MC reconfiguration.....	3-6
Table 3.2 QoS Management Mechanism .....	3-13
Table 5-1 Summary of XMIToPLCopen tranformation .....	5-4
Table 5-2 General structure of MCid_ <i>Control</i> program .....	5-12
Table 5-3: General structure of MCid_ <i>Serialize</i> and MCid_ <i>Deserialize</i> programs.....	5-15
Table 6-1 critical intervals of Station 1.....	6-2
Table 6-2 recovery time for each MCs .....	6-11
Table 6-3 Re-distribution algorithm and reconfiguration times .....	6-12



# **1 INTRODUCTION**

---





## 1.1 Motivation

In recent years, there has been an increase in the investment effort from public institutions to reinforce or recover the manufacturing industries. This sector offers a suitable opportunity for driving innovation, economic growth and job creation. Initiatives such as Factory of the Future in the European Union, Industrie 4.0 driven by the German Federal Government and Advanced Manufacturing launched by the US Government are clear examples. All these initiatives pursue the implementation of high-tech manufacturing processes based on the use of adaptive and smart manufacturing equipment and systems, aiming at automating, controlling and optimizing the processes, ensuring plant availability while providing high quality production with zero defects.

Traditionally, manufacturing systems use hierarchical control structures, which concentrate the processing power of a shop-floor control under one central node. A centralized automation may perform well in terms of production, but respond inadequately under changes in conditions, or with respect to scalability and unpredictability issues. These monolithic, rigid control structures are insufficient to meet the flexibility, robustness, reconfigurability and responsiveness requirements identified as a must by the previous initiatives. As a result of this, new manufacturing systems are being designed with a decentralization and distribution of processing power over several entities, which allows the introduction of mechanisms like dynamic reconfiguration as a way of fulfilling these new requirements.

The introduction of dynamic reconfiguration mechanisms enables a manufacturing system to switch as quick and cost-effectively as possible from one configuration to another while continuing with its normal operation. These changes are performed in response to new production demands and/or unpredictable events, like failures or disruptions.

In the field of industry automation, it is possible to distinguish between different uses of the reconfiguration concept:

- *Product reconfiguration* understood as the flexibility to change or modify the final product.
- *Schedule reconfiguration* in the sense of being able to change the order of execution from the different operations in a plant in order to improve the efficiency or productivity. There are different proposals aiming at achieving these goals that differ on how they perform the reconfiguration: changing the workload of machines, the material handling, the operational time or the operation sequence, among others. This reconfiguration can also be used for other purposes like reducing global energy consumption or avoiding previously detected situations, such as failures or conflictive operations.
- *Machine operation reconfiguration*. In this case, the goal is to modify the functionality of a machine in order to perform other type of operations.
- *Control System Reconfiguration* understood as the ability to relocate the different functionalities over the distributed control system. This allows to cope with controller failures, network failures or to optimize certain Quality of Service (QoS) criteria (such as workload or energy consumption).

From the analysis of the literature it can be concluded that most of the works present smart solutions for ensuring specific QoS (such as production optimization, process fault tolerance, controller failure tolerance or workload balance, among others) by offering an ad-hoc solution for that concrete issue. However, looking at the strategies and mechanisms they propose, it is possible to find similarities, meaning that it might be possible to integrate the strategies within a single reconfiguration architecture providing mechanisms for dealing with different QoS. This work focuses on providing a generic architecture that can be customized to assure different QoS during the system operation.

## 1.2 Research Goals

The QoS requirements of a production system are also known as non-functional requirements. These requirements can be seen as properties that make the product more attractive, usable, precise, safe or reliable. In other words, the non-functional requirements do not modify the product functionality. That is, the functional requirements remain the same no matter the non-functional requirement is, but they add desirable characteristics to the final product, such as reliability, availability, power consumption, task allocation, response time, among others.

Having in mind that a flexible manufacturing system may have to exhibit a sub-set of non-functional properties, the main goal of this work is to propose a **generic management architecture** that could be **customized** for meeting the particular set of QoS requirements of the automation production system. Ideally speaking, this architecture should offer non-intrusive mechanisms that work transparently to the application and that could be easily customized and/or extended to meet the specific QoS requirements of any application.

To guide the design and development of the management architecture, a set of partial objectives has been defined:

- ❖ Identification of QoS requirements that automation production systems typically must meet at run time.
- ❖ Analysis of the information needed to measure QoS demands as well as the actions to be done in order to recover QoS in case of non-fulfillment.
- ❖ To propose a generic architecture that manages the execution of the automation production system offering mechanisms for detecting loss of QoS and QoS recovery.

- ❖ Model-based tool support for developing flexible automation systems to be executed under the control of the run-time platform that implements the architecture.

## 1.3 Structure

Once the motivation and objectives of this research work have been stated in this **first chapter**, the **second chapter** is dedicated to analyse the state of the art divided in two main blocks: 1) dynamic reconfiguration strategies proposed for automation systems and 2) modelling technologies that help in the definition and generation of a reconfigurable automation control system. The chapter conclude with an assessment of what is currently available from which the research challenges are derived.

**Chapters three, four and five** constitute the main contribution of the work. Jointly offer the concept and implementation of a generic run-time platform architecture that can be customized for assuring several QoS requirements in production automation systems.

In particular, in **chapter three** the generic architecture is proposed, describing its components, the offered mechanisms and the information model managed by the architecture.

**Chapter four** is dedicated to the implementation of the architecture as an agent-based platform (build over the JADE framework) that manages run-time entities for measuring QoS and making reconfiguration decisions in case of non-fulfillment.

On the other hand, **chapter five** focuses on a model-based framework consisting of the information model implementation based on the Automation ML (AML) standard that supports the development of the flexible automation production systems.

Finally, **chapter six** is dedicated to the validation of the framework through a case study, as well as to the research assessment.

The last chapter, **chapter seven**, outlines the most important conclusions and contributions of the work and future research lines are commented.



## **2 RELATED WORK**

---





## 2.1 Introduction

As commented in Chapter 1, there exists a growing interest in obtaining more competitive manufacturing systems. The guidelines for this evolution have been defined by the US government, the European Union and the German Federal Government in the projects Advanced Manufacturing (Science & Council, 2016), Factory of the Future (European Commission: Research and Innovation, 2013) and Industrie 4.0 (Blanchet et al., 2014), respectively. All 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 equipment and systems (Association, 2012).

In the literature, three concepts that guide the flexibility of a manufacturing system can be found:

- *Restart(ability)*: to cause a computer program to resume execution after a failure, using status and results recorded at checkpoint (Standards Board, 1990).
- *Recovery*: recovery typically includes two phases, error correction and restart, where “correction” is the process of removing the original problem (the fault) and correcting its manifestation (the error), and “restart” is the process of moving the system to a normal state (Andersson et al., 2011).
- *Reconfiguration*: provides the ability to switch from one configuration to another by creation, removal, replacement and migration of elements (Wegdam et al., 2003).

In the realm of manufacturing systems, the reconfiguration is performed when a change of the software or the hardware is produced either by the user, an automatic process or an external entity. Normally, these changes are brought about by an

update of the system (i.e. addition of new software components, incorporation of new control equipment or machinery, and so on) or a contingency for an event (i.e. failures of a hardware component, a machine or a controller, error in the execution of the software, or changes in the production) (Brennan et al., 2008).

Current works use reconfiguration to provide self-adaptive automation systems, which are able to automatically modify themselves in response to changes in the operation environment (Oreizy et al., 1999; Kephrt & Chess, 2003).

Self-adaptive systems provide so called self-management properties like self-configuration, self-healing in the presence of failures, self-optimization, and self-protection against threats (Huebscher & McCann, 2008; Kephrt & Chess, 2003). For achieving adapting behaviours, basic system properties are self-awareness and context-awareness (Salehie & Tahvildari, 2009). The concept of self-awareness describes the ability of a system to be aware of itself, i.e., to be able to monitor its resources, state, and behaviour (Hinchey & Sterritt, 2006). Context-awareness means that the system is aware of its operational environment, the so called context (Schilit et al., 1994).

In this context and according to different reviews and surveys (Leitão et al., 2016; Krupitzer et al., 2014; Leitão et al., 2013; Wang et al., 2012; Vrba et al., 2011; Shen et al., 2006) the concept of ‘reconfiguration’ is used to represent different situations: when applied to *product*, it is understood as the flexibility to change or modify the final product. *Schedule reconfiguration* commonly denotes the ability to change the execution order of a plant operation in order to improve the efficiency or productivity (Nouri, 2015; Urban & Chiang, 2016) or overcoming machine failures (Legat & Vogel-Heuser, 2014). Sometimes it also refers to the modification of a machine functionality in order to perform another type of operations (*machine operation reconfiguration*) (Ribeiro et al., 2015; Rocha et al., 2014). Finally, *control system reconfiguration* is understood as the ability to relocate the different functionalities over the distributed control system, as a way of optimizing controller performance (Botygin &

Tartakovsky, 2014) or assuring the execution despite controller or network failures (Merz et al., 2012; Streit et al., 2014).

The following sections present different approaches that tackle the reconfiguration in manufacturing systems.

## **2.2 Manufacturing System Reconfiguration**

Many of current manufacturing control applications follow the IEC 61131-3 standard (International Electrotechnical Commission, 2003). This standard, however, has been optimized for the use of a centralized control system and global variables, hindering the development of distributed control systems and the integration of reconfiguration mechanisms (Vyatkin et al., 2005).

Nevertheless, the IEC 61131-3 is not the unique standard defined for manufacturing control applications. There also exists the IEC 61499 standard (Commission, 2004), which was developed in order to cope with distributed systems. It focuses on an extended definition of Functional Blocks (FB) and it provides the requirements for software tools to support the specification, analysis and validation of distributed control systems.

The IEC 61499 has been widely accepted by the academic community; a big number of publications has been produced and a debate on pros and cons is active (Thramboulidis, 2006) and (Thramboulidis, 2013). Interestingly, the standard has not been accepted by the industry (Basile et al., 2013). This is due to many reasons including the absence of support by the currently dominating tools and environments in industry and the absence of a variety of new mature tools and run-time environments to support the new standard. The lack of “mature engineering tools, reliable embedded control hardware, proven design methodologies, and trained engineers” is considered in Vyatkin, (2011) as the main barrier that prevents practitioners from using the IEC 61499. Even if the IEC 61499 standard is not widely

used in the industry, many reconfiguration works still use it as a basis for their approach.

In fact, current reconfiguration approaches are based on the collaboration of control applications implemented in any of the standards, the IEC 61131 or the IEC 61499 and the so called Multi Agent Systems (MAS) (Wooldridge, 2009; Ferber, 1999). This latter is a computational paradigm introduced in the distributed artificial intelligence field. The MAS is characterized by the decentralization and parallel execution of activities based on autonomous agents. MAS solutions replace the centralized control by a distributed functioning where the interactions among agents lead to the emergence of an “intelligent” global behaviour, being able to react and adapt to condition changes without external intervention (Wooldridge, 2009). The decentralization of control functions over distributed autonomous and cooperative agents facilitates modularity, autonomy, flexibility, robustness and adaptability. An agent can be defined as an “autonomous component that represents physical or logical objects in the system, capable to act in order to achieve its goals, and being able to interact with other agents, when it does not possess knowledge and skills to reach alone its objectives” (Leitão, 2009).

Besides, an agent can sense its environments and make decisions according to its internal behaviour, knowledge and objectives. Aiming to address the emerging challenges of self-optimization, self-healing and self-organization, an agent is required to provide a set of self-X properties (Onori et al., 2011; Leitão, 2008; Bousbia & Trentesaux, 2002). All these properties given, the agents play a key role in the implementation of self-reconfigurable systems.

As presented in (Babiceanu & Chen, 2006; Leitão, 2009; Leitão et al., 2013; Vrba et al., 2011; Leitão et al., 2016), over the last three decades several agent methodologies and architectures have been implemented for manufacturing rescheduling and reconfiguration. Duffie and Piper (1986), for instance, were the first ones to discuss and to introduce the heterarchical control approach, using agents to represent

physical resources, parts and human operators, and implementing scheduling oriented to the parts.

The product-resource-order-staff architecture (PROSA) (Brussel et al., 1998) is, meanwhile, one of the first holonic reference architectures used for the rescheduling of manufacturing systems. The introduction of holons allows optimizing the use of the resources in system. A holon, as Koestler devised the term, is an identifiable part of a (manufacturing) system that has a unique identity, yet is made up of sub-ordinate parts, and in turn is part of a larger whole (Koestler, 1969). The PROSA architecture is based on three types of basic holons that represent the products, orders and resources of the system. These holons are implemented in a specific agent in a MAS. The resource holon contains the control logic and the information of a certain resource. The product holon holds the process and product knowledge, as well as all information about the product. The order holon represents the tasks in manufacturing systems. Additionally, the architecture defines staff holons, whose mission is to assist and advise the other holons. The collaboration of the order and resource holons is used to accommodate the client demands. In addition, order holons use negotiation techniques to ensure fast and reliable production. On the other hand, the main aim of resource holons is to maximize the return on the execution of their services.

The ADACOR (ADaptive holonic CONTROL aRchitecture for distributed manufacturing systems) (Leitão & Restivo, 2006) manufacturing control architecture addresses the agile reaction to emergencies and changes. It increases the agility and flexibility of the system when presented with volatile environments, which are characterized by the frequent occurrence of disturbances. For this purpose, it introduces an adaptive control approach that evolves in time to combine the global production optimization with the agile reaction to disturbances, where the supervisor entities and the self-organization and learning capabilities associated to the holons are the key roles that support the dynamic evolution and reconfiguration of the organizational control structure. Barbosa et al. (2015) present ADACOR2, which extends the evolutionary

process allowing the architecture to find a multitude of dynamic reconfigurations, instead of the two defined in ADACOR.

The CoBASA (Coalition Based Approach for Shop floor Agility) architecture (Barata & Camarinha-Matos, 2003) focuses on the shop floor re-engineering, using agents to represent the physical components which are aggregated into consortia regulated by contracts, achieving agility in the shop floor life-cycle. Lastra (2004), on the other hand, proposes the actor-based assembly system (ABAS) architecture to develop reconfigurable assembly systems in an easy way.

Zhou et al. (2007) also proposed a three level MAS that coordinates the product order by distributing the work to the different cells and machines. The System Optimal Agent (higher level) distributes the production order by means of a negotiation with the cell agent (intermediate level). Every time a new order is assigned, the Cell Agent generates a new Job Agent in its lower level. These Job Agents negotiate with the Machine and Material Handling Device agents to determine the distributions of the operations.

Other approaches use MAS to optimize the production process based on production and order information. Morenas et al. (2012), for instance, use the information regarding the client, the deadline, and the arrival date, to define the priority of each product as well as to optimize the production. On the other hand, Nouri (2015) uses other parameters such as machine workload, material handling, operational time and operation sequence of the parts, among others.

The ARUM project (Marín et al., 2013) combines MAS and SOA (Service Oriented Architecture) to develop knowledge-based applications that support re-scheduling to respond faster to unexpected events (sudden increase of process instances, off-sick staff, broken machines, etc.) in ramp-up production of complex and highly customized products, such as airplanes or shipyards. The optimisation process considers the available resources, timing, constraints, and unexpected events.

The PRIME project (Rocha et al., 2014) proposes a MAS framework to enhance assembly systems with standardized plug and produce process and control solutions to allow rapid reconfiguration and deployment of new machines. During the plug-in process, a Skill Management Agent (SMA) extracts the skills of the new machine, which are later used by the Production Management Agent to reschedule the production. Plug and produce is also provided by the IDEAS project (Ribeiro et al., 2015) that focuses on the application of key enabling technologies, and particularly MAS, to enable instant/plug and produce deployment of modular equipment without reprogramming.

Works like SOCRADES (Colombo et al., 2010; Colombo et al., 2015) focus on the use of service-oriented architecture and agent systems to provide cooperation among different devices in order to achieve a specific goal. The collaborative automation units are able to expose and/or consume services, for each production scenario in a defined production domain, e.g., electronics assembly, manufacturing, continuous process, etc. A collaborative unit can be a simple intelligent sensor or a part/component of a modular machine, a whole machine and also a complete production system.

However, rescheduling processes can also be used to deal with machine and operation failures. Legat and Vogel-heuser (2014) present for example a two level MAS that re-schedules a monolithic manufacturing system (one controller) when failure of an operation is detected or when a new order arrives. The high-level agent provides the rescheduling, while the lower level agents oversee the execution of operations and detect possible failures based on operation execution time.

In the realm of component failures, (Schütz, Wannagat, et al., 2013; Wannagat & Vogel-Heuser, 2008) presents the use of multi agent systems to assure dependability of the production in presence of sensor failures. Agents are dedicated to physical equipment, e.g. machine parts, to be controlled (Schütz, Wannagat, et al., 2013). Agents exchange their knowledge about related sensor data in real-time to calculate

virtual sensors in case of sensor degradation or faults; this serves to increase machine availability by operating with the virtual sensor with lower precision instead. Thus, automation agents are closely related to the physical layer of a plant.

The previous works provide the reconfiguration based on the assumption that each machine provides a fixed number of control functionalities, which can be activated to perform an operation of the production plan. However, the reconfigurations presented in manufacturing systems also extend to the modifications that affect these functionalities.

In this context, (Olsen et al., 2005) propose a MAS architecture that reconfigures a IEC 61499 control system by means of adding, removing or re-connecting the Functional Blocks (FB) of the application. Each FB is extended with mechanisms to monitor and control the configuration of the system. These FBs communicate with a series of external agents that collaborate to determine the new configuration. The reconfiguration actions to cope with the different situations, like machines or controller failures, must be previously defined by the user. Other works like Lepuschitz et al. (2011) provide more flexibility to this concept by dynamically generating the new configuration of the control application. In a same manner, Yang et al. (2013) use these functionalities to provide a plug and play functions for a distributed HMI.

Works like (Khalgui & Mosbahi, 2010) and (Zhang et al., 2015) extend the concept of adding and removing FB to allow the relocation of FBs into other controllers. This relocation is based on moving the code of the FB as well as its execution data (its state). This type of reconfiguration is known as *stateful reconfiguration*. These authors propose an external MAS that provides the relocation of the FBs.

In a similar manner, the eCEDAC approach (Schimmel & Zoitl, 2011) implements the reconfiguration through the so-called Reconfiguration Execution Control Function Block, which is able to generate a copy of an FB in some other controller, as well as

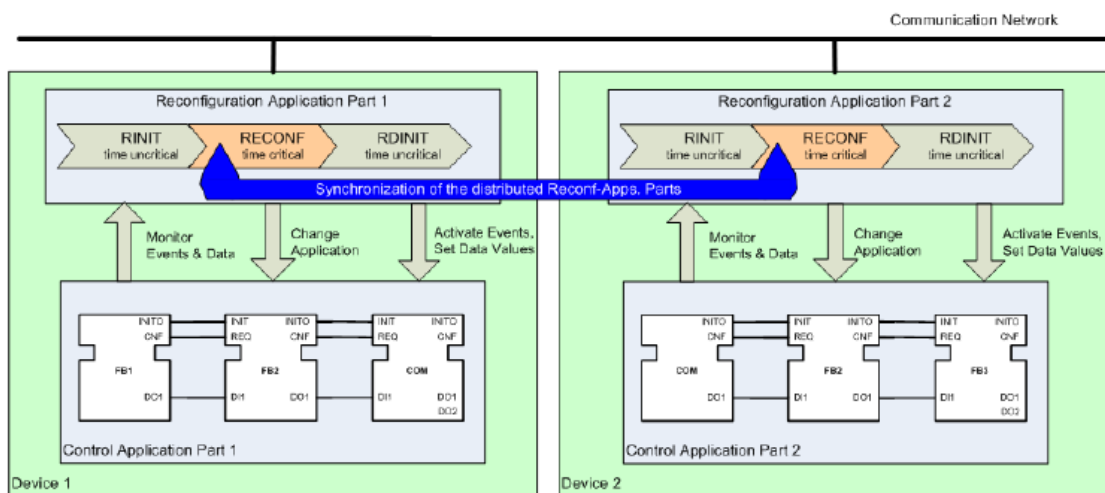


restoring their data, rearrange their communications and remove the old FB. This process is divided in three phases:

(1) RINIT, in which the initial preparation of the necessary elements is done. Here the instances of the FBs and their communication FB are generated (in stop state) as well as the event and data connections associated to each one of them. This task is not time critical.

(2) RECONF, which encapsulates all operations necessary to change from the old state to the new one. All FBs affected by the online changes must be stopped. Afterwards all events and data connection from old FBs can be deleted and the state of these FBs is transferred to the corresponding FBs. Finally, the FBs can be restarted and the application can resume. This sequence is time critical since it denotes the switching from the old application to the new one.

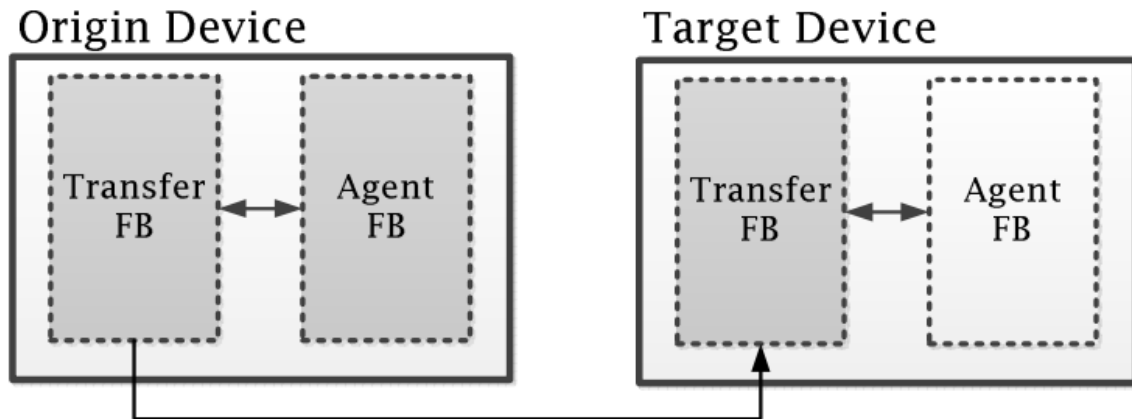
(3) RDINIT, which is a non-time critical task responsible for cleaning up after reconfiguration by deleting the old FBs (see Fig. 2-1).



**Fig. 2-1 eCEDAC-Approach for Distributed Online Change** (Schimmel & Zoitl, 2011)

The previous approaches deal with an external entity, which provides the reconfiguration mechanism. However, works like the one presented by Yan and

Vyatkin (2013) define new types of FBs which contain the reconfiguration mechanism. They present the Agent FB that executes control code and the Transfer FB that is an extended communication FB that can move the code and state of the Agent FB to the new controller.



**Fig. 2-2 A Transfer FB moving an Agent FB from the origin device to the desired target device (Yan & Vyatkin, 2013)**

Nevertheless, there are works like (Strasser & Froschauer, 2012) in which the FBs are duplicated in different controllers. In this approach, one of the controllers acts as a master, being in charge of storing the state of the applications and monitoring other slave controllers. The slave controllers are the ones that are running the different control logics (FBs). In the cases of a controller failure, the master selects a new slave controller and restarts the execution of the FBs from their last execution state. In the same manner (Merz et al., 2012) and (Streit et al., 2014) provide a reconfiguration based on controller and/or network failures in which the master controller also executes part of the control logic.

## 2.3 Model Driven Engineering in Manufacturing System

The introduction of reconfiguration mechanisms increases the complexity of the automation system in terms of size, functionality and distribution, making the design

and development processes more complex as well. 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 (Selic, 2003). These techniques have been applied to provide comprehensive system description, characterization of QoS requirements and even generation of the system implementation.

In the realm of industrial automation, some authors have used the Unified Modelling Language (UML (Booch et al., 2015)) to describe IEC 61131 control systems (Estevez et al., 2005; Hästbacka et al., 2011) and IEC 61499 control systems (Thramboulidis et al., 2006; Vyatkin & Hanisch, 2009). More recently, Systems Modeling Language (SysML, 2007) has been adopted (Thramboulidis, 2011; Schütz, Obermeier, et al., 2013; Jamro, 2014). Other works also use model techniques and design patterns (Fay et al., 2015), aspects (Wehrmeister et al., 2014) or Petri-nets (Basile et al., 2013). All of them integrate model-based techniques into the development process.

On the other hand, for the concrete case of IEC 61131 standard, PLCopen, which is a vendor- and product-independent worldwide association, has defined a common representation format for defining the IEC 61131-3 software model (Marcos et al., 2009; Van der Wal, 2009). Although the aim was to provide interoperability among programming tools, it also provides a model for describing the application software of automation systems, allowing easy code generation.

Modeling techniques have also been used to support the development of the overall automation system. Thramboulidis (2010) propose the “3+1” architecture in which the system is designed from three different domains (software engineering, mechanical engineering and electrical engineering) using specific domain tools. The “+1” model relates the other three acting as a link to form the whole system.

Following this ideas, works like Estevez et al., (2005) propose a MDD approach that uses domain languages based on UML profile definition. From the complete UML model the software architecture is automatically generated in PLCopen XML format, importing the functional code from PLC libraries. In the same manner, the approach

presented in (Vogel-Heuser et al., 2014) proposes SysML-AT, a specialized profile that extends SysML for defining the hardware and software of the automation & control system. In this case, the modeling approach is integrated into the commercial tool CODESYS.

Other works use models to extend the definition of the system elements (FB, machine, controller, component, among others) in order to allow system reconfiguration. For example, the research project *Functional Application Design for Distributed Automation Systems* (FAVA) (Fay et al., 2015; Anon, n.d.) 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 deployment of FBs in system nodes. Although in this work the deployment is defined at development time, authors point out that it could be used at runtime in order to launch system reconfiguration.

Vogel-Heuser and Rösch (Vogel-Heuser & Rösch, 2014) introduce 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 calculated one (Schütz, Wannagat, et al., 2013).

On the other hand, the AMoDE-RT approach (Wehrmeister et al., 2014) presents an aspect based characterization of the non-functional requirements of the functional components, allowing the run-time monitoring of these requirements and the reconfiguration in case of non-fulfillment. The approach is applied in (Binotto et al., 2013) to embedded control systems.

The SOCRADES project (Cândido et al., 2011) uses a model based definition of the functionalities of a machine. This information is used by a holonic architecture to distribute the job to machines at runtime. Another interesting work is presented in (Legat et al., 2013), where both, the operations of machines and the operations

required by a product are modelled. This information is used a novel modelling approach which enables describing what a plant is able to do. This information is used to automatically derive an optimal operation sequence. A continuation of the previous works is presented in (Bergagård, 2015), which deals with the calculation of the control state from which the production can be restarted.

## 2.4 Conclusions

After researching many works dealing with reconfiguration, some conclusions have been reached regarding flexible manufacturing systems.

The use of MAS technologies has simplified the implementation of reconfiguration mechanisms, since multiple agents with specific functionalities can be generated; for instance, machine agents, optimization agents, job agents, management agents, supervisor agents, reconfiguration agents and so on. The interaction of these different agents can be used to ensure the optimization and availability of the system.

As a summary, Table 2-1 groups the previous works based on different characteristics of the reconfiguration mechanism:

- **Context-awareness:** are those works that take into account the state of its operational environment in order to launch or not a reconfiguration decision.
- **Stateful reconfiguration:** these works launch the system from the last known state following a reconfiguration.
- **Reconfiguration Trigger:** present the different types of non-functional requirements that are managed by this reconfiguration mechanism.
- **Language:** standard and technologies used in the definition of the reconfiguration mechanism.
- **Tool support:** provide mechanism for defining the reconfiguration scheme during the design phase.

**Table 2-1 Control System Reconfiguration Approaches**

Works	Context awareness	Stateful Reconf.	Reconfiguration Trigger					Lang.	Tool Support	
			Production Resource Optimization	Machine or Component Failure	Machine operation	Controller Optimization	Network Failure			Controller Failure
PROSA; ADACOR; ADACOR2; CoBASA; (Zhou et al., 2007); (Morenas et al., 2012); (Nouri, 2015); PRIME; IDEAS;	X		X						MAS	
ARUM;	X		X						MAS; SOA	
ABAS;	X		X						MAS	X
SOCRADES;	X		X						MAS; SOA	X
(Legat & Vogel-Heuser, 2014); (Legat et al., 2013); (Bergagård, 2015);	X		X	X					MAS; IEC 61131	X
(Vogel-Heuser & Rösch, 2014); (Schütz, Wannagat, et al., 2013);				X					MAS; IEC 61131	X
(Olsen et al., 2005); (Lepuschitz et al.,					X				IEC 61499	

2011)										
(Khalgui & Mosbahi, 2010); (Zhang et al., 2015); (Schimmel & Zoitl, 2011); (Yan & Vyatkin, 2013)		X				X			MAS; IEC 61499	
(Strasser and Froschauer 2012);		X						X	C	
(Merz et al., 2012); (Streit et al., 2014)		X					X	X	C	
FAVA;						X			IEC 61499	X
(Wehrmeister et al., 2014); (Binotto et al., 2013);						X		X	C and Java	X

As it can be seen, most of the works that deal with changes of the control system emphasize the importance of a *stateful reconfiguration*, in order to continue the execution of the control logic. However, the *stateful reconfiguration* presented by these works does not consider the effect that the reconfiguration process has on the production, i.e. the cases of reconfiguration in which the whole state of the production system is not known, commonly associated to faults. These approaches directly recover the execution from the last known state of the control logic, without taking into account of the physical system before the reconfiguration process.

The state of the physical system can generate situations in which the reconfiguration cannot be performed. So, before launching a reconfiguration, it is necessary to determine the actions to be done on the production system before and after the reconfiguration takes place to prevent inconsistency states. Lepuschitz et al. (2011) and Bergagård (2015) presents a solution to this problem in the case of process instrumentation failures by defining two sequences of operation that surround the reconfiguration process (before and after); this helps in the preparation of the system for the reconfiguration, and prepares the machine for the new configurations respectively.

Another big drawback of the current reconfiguration approaches has to do with the QoS they ensure and the implementation of the reconfiguration mechanism. As far as authors know, these works ensure specific QoS (production optimization, process fault tolerance, controller failure tolerance or workload balance, among others) by offering a custom solution to the concrete issue and most of them are mainly based on implementation languages that are not commonly used in the factory automation area. While the re-scheduling works use industrial standards for the control of the machine, the works on control logic reconfiguration mechanism use the IEC 61499 standard and C programming languages.

The present work takes these ideas and goes beyond integrating them into a generic architecture that provides QoS management, thus achieving production unaware reconfiguration for IEC 61131 automation production systems.







## **3 QoS MANAGEMENT ARCHITECTURE**

---



## 3.1 Introduction

Ensuring non-functional requirements of a manufacturing system are responsibility of the control and automation system and deal with very different aspects, such as availability, energy efficiency, system performance or production system flexibility. In this context, the present work proposes a customizable and extensible architecture for assuring QoS requirements automation production systems, by means of dynamic reconfiguration.

This chapter focuses on identifying the key concepts and mechanisms needed to deal with:

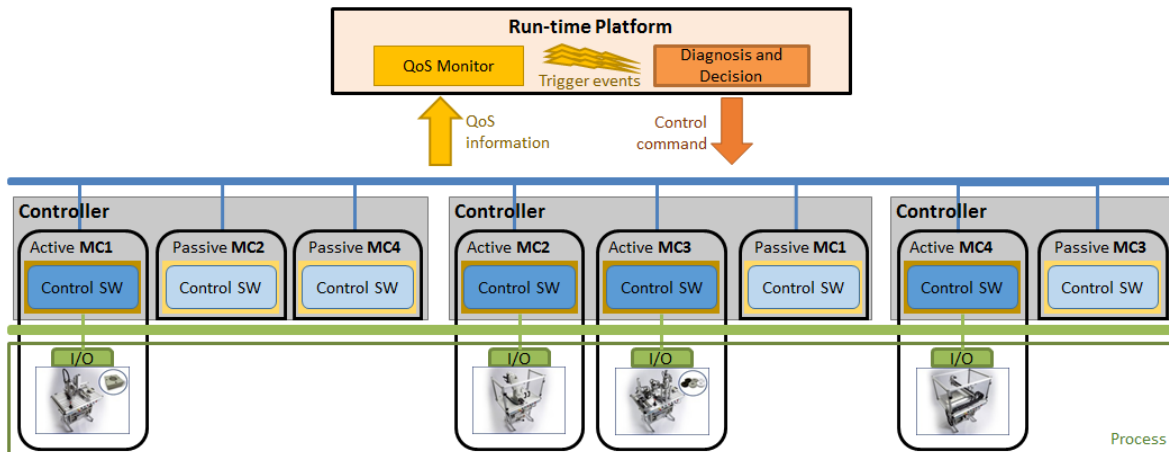
- Monitoring the fulfilment of a QoS and detecting QoS loss.
- Recovering the QoS, if possible, by means of control system reconfiguration.

It is important to remark that the monitoring and recovery processes depend on the type of QoS. The goal of the architecture is to provide the basic components and information model from which the different processes can be implemented.

The following sections present the key concepts of the architecture, describing its main components and their responsibilities. The information managed by the generic architecture is formalised and structured based on the knowledge about the production process acquired during the design phase. This information defines the relevant operating conditions of the production automation system.

## 3.2 General Scenario

Fig. 3-1 illustrates the proposed flexible automation production process:



**Fig. 3-1 Flexible Automation Production System**

It contains the set of relevant elements that are described in detail in the following sections. A flexible automation production system comprises:

- i. A set of *distributed controllers*, in charge of controlling the process. From now on, it is assumed that they can access different sets of I/O via a network controller, i.e., every controller may control different parts of the process.
- ii. A set of the so-called *mechatronic components* (MCs). A mechatronic component is an atomic part of the plant that can run autonomously coordinated with others through logical inputs and outputs. In this work a MC is constituted by:
  - A part of the process (mechanics, electrics and electronics).
  - A set of instrumentation and control hardware (sensors, actuators and their corresponding controller I/O).
  - A consistent piece of control software that can reside in a number of controllers. At run-time, every MC will only be active in one controller.

- iii. A run-time-platform that tracks the overall system operation detecting QoS losses, acting consequently to meet the requirements as soon as possible. It comprises two main components:
- The so-called *QoS monitor* component, which is in charge of verifying the fulfilment of the QoS at runtime. The monitor might have multiple functionalities if multiple QoS have to be managed. For instance, monitoring Performance means tracking workload, while Availability monitors the liveness of controllers.
  - *Diagnosis and Decision* component. It is responsible for analysing the *loss of QoS* triggers, combining the different trigger events to make a decision based on the current state of the manufacturing process. As a result, it issues control commands to the MCs involved in the reconfiguration process. Such commands de-activate / activate MCs in the indicated controllers, sometimes after executing pre-determined actions in case of controller failure.

## 3.3 Production system information model

The information model of the production automation system comprises the information model of every element in Fig. 3-1. The following subsection presents the description of each of these elements.

### 3.3.1 MC information model

The concept of mechatronic component (MC) was first proposed by Thramboulidis (2005) as a component having a physical electromechanical part, and a software part. Pang and Vyatkin (2010) have further divided the definition of a MC into a mechanical part (physical functional device with sensors, actuators, and electronic circuits), an embedded control device (computing device with interfaces to the

sensors, actors, and networks) and a software components: (a set of data and control logics).

Other works like separate the MC from the embedded controller in which it runs. Lüder et al. (2010) define mechatronical units as containers of relevant information related to its mechanical part, electronic part, and software part. In this work, the concept is extended to represent run-time information: In particular the situations in which the current state of the mechatronic component can be derived from the current values of the controller variables. Besides, the software and the physical part can be dynamically assigned to different control devices.

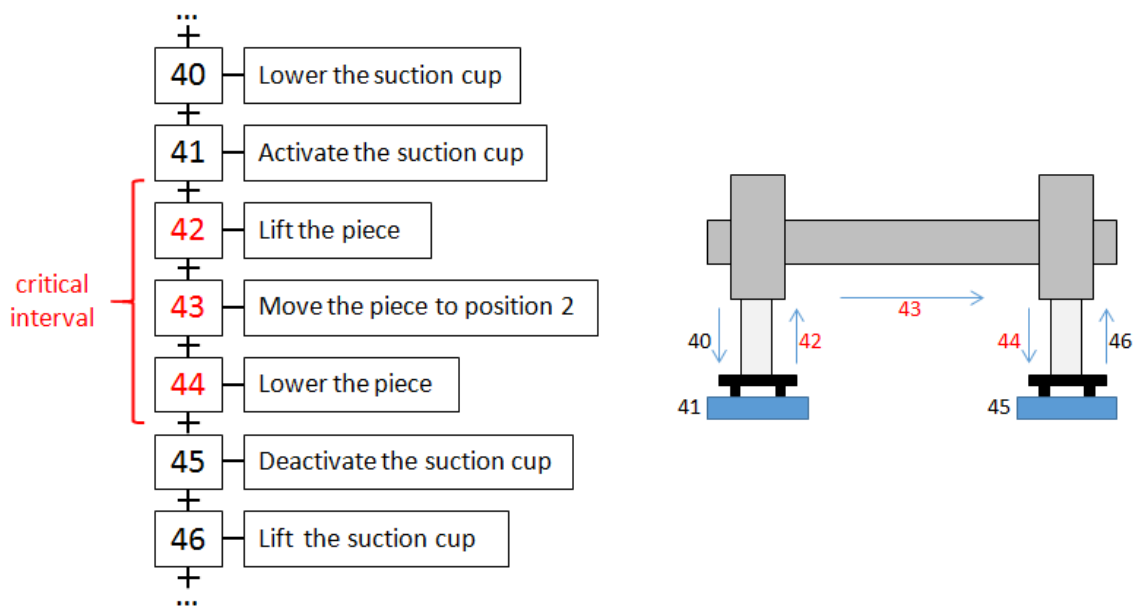
This information is fundamental to enable dynamic reconfiguration. The use of dynamic reconfiguration for assuring some type of QoS (such as, for instance, system availability or efficiency) requires not only to monitor such QoS (using specific techniques such as for instance heart beat or work load, respectively, in all controllers) but also assuring that the reconfiguration is possible and the manufacturing process will not suffer unpredictable effects. For instance, if the work of controllers is unbalanced and it is necessary to transfer control tasks among controllers, the transference of workload must be performed at instants in which the production process not be affected. This means that the operation state of the process must be known by the controller.

Depending on the manufacturing process and the design of the automation production system, it is possible to distinguish the following run-time situations:

An operation state of the MC is *non-critical* if it may be activated directly in another controller (after de-activation in the current controller or after a controller failure) having as initial state the last known state of the MC. On the contrary, *critical* states are those not fully represented by the current values of the software variables of the MC, preventing activation/de-activation of the MC in order to avoid unpredictable process behaviour.



As an example, let us analyse the different critical states present in the movement of a piece using/by a crane. In this case, the critical states are related to the lifting, transporting or placing of the piece. These states are defined as critical since the lost of the control during these operations can mean the release of the piece, which prevents the continuation of the production. These states (lifting, transporting and placing) are defined as critical interval during which the corresponding MC should not be de-activated.



**Fig. 3-2 Example of critical interval**

During the design of the automation production system, it is possible to identify the critical states of the process from the point of view of the control system. Thus, it is possible to know in advance if MCs can be activated and de-activated at runtime.

Table 3.1 summarizes when a MC reconfiguration is possible, attending at the MC state. Note that recovering a MC from a controller failure is a special case in which a critical state has two variants: those cases in which even the state is not completely known, the MC could be re-started from a checkpoint state and those in which the state is unpredictable and thus the MC cannot be re-started.

Table 3.1 MC reconfiguration

MC State	Reconfiguration (MC de-activation /activation)	Controller failure recovery
Non-critical	Always possible (from last known state) (reconfiguration enabled)	Always possible (from last known state). (reconfiguration enabled)
Critical state	Wait until non-critical state is reached (reconfiguration disabled)	MC can be launched from a previous known state ( <i>checkpoint</i> ), possibly after executing recovery actions. (reconfiguration enabled)
		MC recovery is not possible. Action: safe stop. Operator warning. (reconfiguration disabled)

This extension of the MC concept leads to the following formulation:

The overall control system is composed by a set of MCs that in turn comprise a set of elements containing relevant information about the operating part of the system:

$$\begin{aligned}
 Sys &= \{MC_i\}, i=1..n \\
 MC_i &= \{types_i, main_i, S_i, VI_i, VO_i, \Sigma_i, \Psi_i, \delta_i\}
 \end{aligned} \quad (1)$$

where:

- $types_i = \{D_i, P_i\}$  represents the set of derived data types ( $D_i$ ) and Program Organisation Units (POU in the context of IEC 61331-3 standard software model) types ( $P_i$ ) that compose the control logic of the  $MC_i$ .
- $main_i \in types_i$  is the POU type of the main program from which the rest of POU types of the  $MC_i$  are instantiated and used.

- $S_i$  is the set of variables corresponding to the  $MC_i$ . The variables are characterized by their name, type, and in the case of inputs and outputs, they are also characterized by its physical address (2).

$$s_i^j = (\text{name}, \text{type}, [\text{address}]); s_i^j \in S_i; \quad (2)$$

*j=1..m state variables*

- $VI_i \subset S_i$  is the set of inputs coming from the external world (process, operator, HMI, ...) that correspond to global variables.
- $VO_i \subset S_i$  is the set of outputs associated to the physical variables related to the physical part of the  $MC_i$  corresponding to global variables.

The rest of elements that compose the MC define the critical states of the production system in which the MC must not be activated / de-activated as this action may lead to an inconsistent state of the MC. As commented above, critical states can be represented as the values of selected variables in  $S_i$ . Note that a controller failure in any of these states will lead to a non-direct recovery.

Critical intervals are defined as the set of critical states that have associated the same recovery method (checkpoint state and/or recovery actions). The critical interval is characterized by:

- i. An expression involving values of variables of the MC state.
- ii. A recovery action and/or a recovery state (checkpoint) needed to resume the execution. Note that a state that not belongs to a critical interval can be directly recovered.

Therefore,

- $\Sigma_i = \{\Sigma_i^k\}$ ,  $k=1..l$  *critical intervals* is the input alphabet that defines all the expressions associated to critical intervals of the set of MCs. Every critical interval has its own expression,  $\Sigma_i^k$ , that allows determining if an execution state belongs to the interval. It is a Boolean expression composed by arithmetic and logical operations of state values following the grammar

$$\begin{aligned} \langle Exp \rangle &::= \langle Var \rangle \langle arithmetic \rangle \langle Value \rangle \mid \langle Exp \rangle \langle logic \rangle \langle Exp \rangle \\ \langle arithmetic \rangle &::= \leq \mid \geq \mid < \mid > \mid = \\ \langle logic \rangle &::= OR \mid AND \\ \langle Var \rangle &\in S \\ \langle Value \rangle &\in \mathbb{R} \end{aligned}$$

- $\Psi_i = \{\Psi_i^k, \emptyset\} / \Psi_i^k = \{\xi_i^k, \lambda_i^k\}$ ;  $k=1..l$  *critical intervals* represents the output alphabet that defines all possible recovery methods. The recovery methods for critical states include an action ( $\xi_i^k \in P_i$ ), a checkpoint ( $\lambda_i^k \in S_i$ ) or both. The checkpoint is the state value from which the execution must be resumed.
- $\delta_i: S_i \times \Sigma_i \rightarrow \Psi_i$  is the output function that assigns to a critical interval an element of the output alphabet.

### 3.3.2 Controller information model

A distribution automation system is constituted by a set of controller equipped with I/O hardware to acquire process data (I) and to act on the process (O). Without loss of generality, in this work it is assumed that the access to the process is performed via a network protocol, which can be re-assigned to a different controller dynamically. i.e. the inputs and outputs of a controller can vary dynamically following a reconfiguration trigger (in the validation of the work in chapter 5 this is implemented, as a proof of concept, by having one unique PROFIBUS master communicating with the rest of the controller in order to update the current value of the I/Os).

Every controller can be described from a hardware and software point of view. The hardware definition follows the recommendations of ISO 15745-3 that allows

describing hardware elements from different views: vendor or manufacturer information; the generic functionality of the device; and the behaviour of the device in the application process. On the other hand, the software of a controller describes the control logic of different MCs of the system, following the IEC 61131-3 standard software model.

According to the IEC 61131-3 standard software model, the software architecture is built upon the following elements: the *Configuration* represents a PLC, the *Resource* provides support for program execution (a CPU or a virtual machine), and the optional concept of *Task* allows the designer to define the execution rate of different parts of the target code (International Electrotechnical Commission, 2003). The *task* is characterized by its priority and period. In this work it is assumed that the MC control is performed by a Program instance of the corresponding MC POU Program.

The POUs are defined by their interface and they could be programmed in any of the programming languages provided by the standard. The Function Block POUs can be used to program the control strategies and they can be used as variables of more complex POUs, for example to define the main program of a MC. Each POU is characterised by its *body* and *formal parameters* (interface).

As commented above the input and output variables of the MCs are part of the *global variables*. The IEC 61131-3 variables are characterized by their name, type, value and physical address (*at*).

The meta-model corresponding to the software view is illustrated in Fig. 3-3.

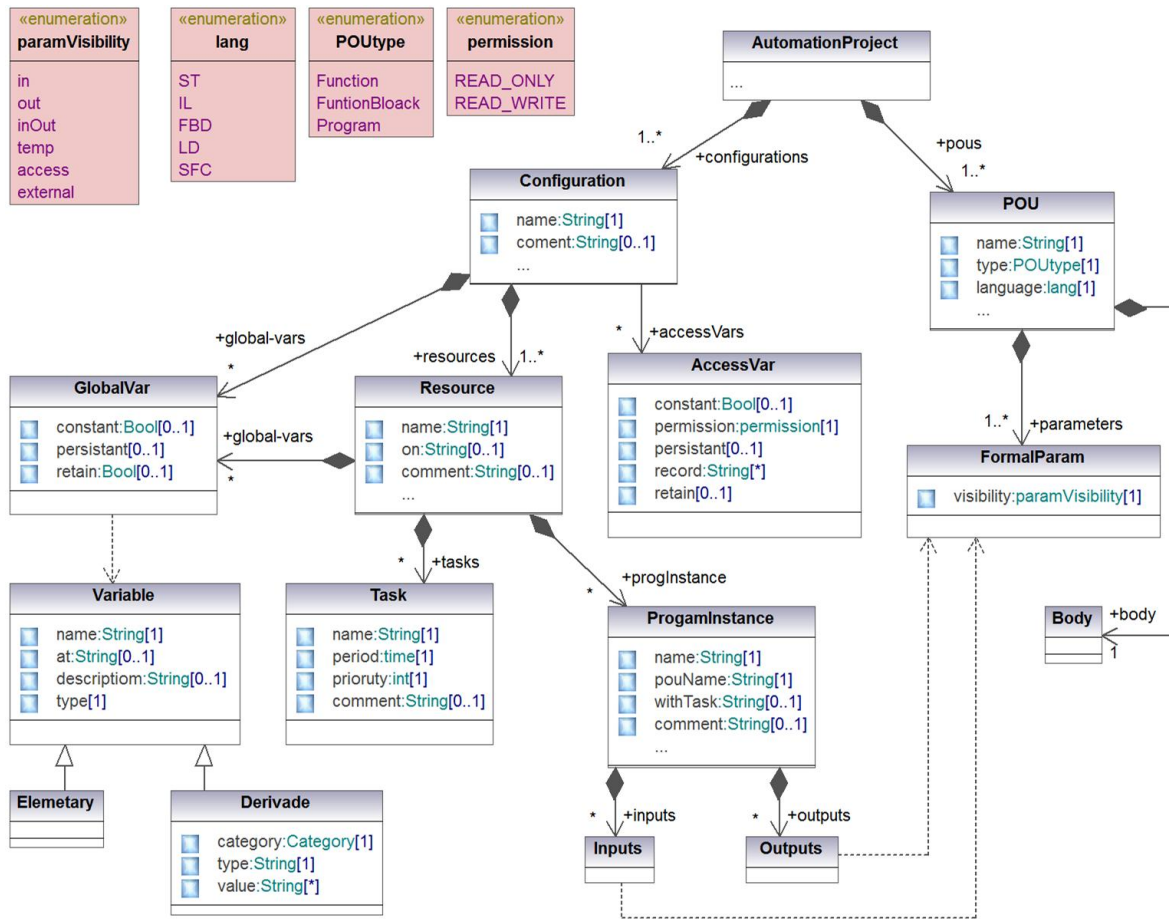
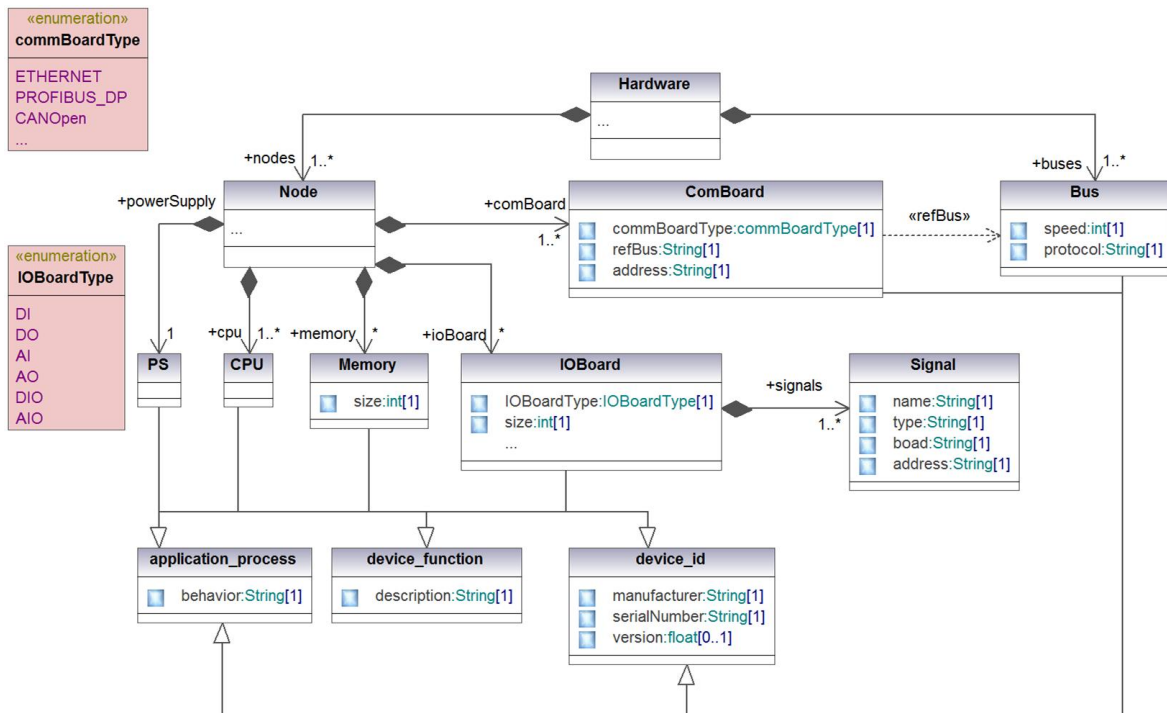


Fig. 3-3 Software View: concepts and their relationships

The hardware view deals with the selection of the equipment needed to start-up the control system. The hardware architecture of industrial control systems uses specific hardware equipment. It commonly consists of a set of network nodes that correspond to industrial controllers (PLCs) or data acquisition devices (nodes) connected through a set of network segments that are called industrial communication systems (buses). The concepts of this domain are illustrated in Fig. 3-4.



**Fig. 3-4 Meta-model of the hardware concepts and their relationships**

Fig. 3-4 depicts the characterization of the devices, taking into account the elements defined by the ISO 15745 part 3 standard (FDCML, 2002), namely: (1) *device identification*: the set of properties that identify the device; (2) *device manager*: the set of properties used to configure and to monitor a device belonging to the application (3) *device function*: description of the intrinsic function in terms of its technology and (4) *application process*: description of the behaviour of the device from the application point of view.

Devices (nodes and boards) are characterized by three types of information: *manufacturer*, *serial number* and *version*. The information needed for the equipment start-up and maintenance is collected in the device manager, for instance, information about I/O signals. *Description* defines the intrinsic function of device's technology. Finally, the application process information is defined by the *behaviour* property.

On the other hand, the different *signals* of an I/O board are characterized by their *name*, *type*, I/O board and physical *address*.

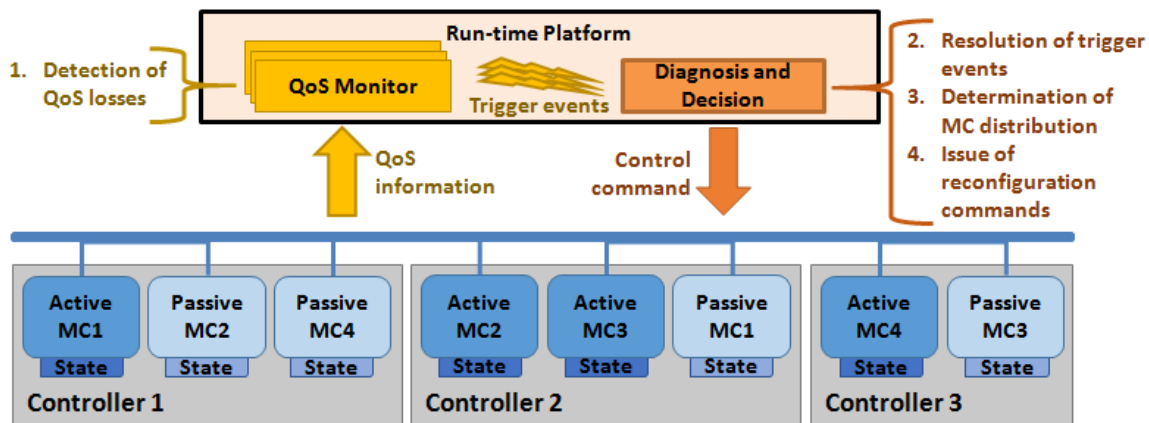
With respect to the software and hardware relationships, configurations and resources will be mapped to the corresponding processing node, and each I/O signal of the hardware architecture must correspond to a global variable in the software architecture.

These models are used in Chapter 4 to automate the generation of the automation projects corresponding to the controllers in the system.

### 3.3.3 Run-time platform model

An important component of the general scenario is the run-time platform which making use of the information model of MCs and controllers, monitors QoS fulfilment, triggering if needed reconfiguration processes.

The run-time platform also manages information about the controllers that are present in the system, the controllers in which each MC is deployed (at least in one) and the controller in which each MC is active. In order to track system QoS, detecting QoS loss and making decisions to recover the system as soon as possible, the management platform comprises as many *QoS Monitor* components as QoS must be assured and one *Diagnosis&Decision (D&D)* component, as Fig. 3-5 illustrates.



*Fig. 3-5 Run-time platform*

Each *QoS Monitor* is in charge of measuring specific QoS, triggering internal events to the *D&D* component in case of QoS loss. *D&D* receives QoS loss triggers and making



use of the MCs current state and system state makes a decision on the actions to be performed to recover QoS. Finally, these actions may imply MC relocation, possibly after the execution of recovery actions. Table 3.2 illustrates the mechanisms for different QoS as well as the use of MC state.

**Table 3.2 QoS Management Mechanism**

<b>QoS type</b>	<b>Monitoring mechanism</b>	<b>Diagnosis and decision goals</b>	<b>Use of the state</b>
Availability	Liveliness	Re-assignment of MCs of failed controller if possible	To obtain the recovery method
Energy efficiency	Workload, N° of controller running	Minimum active controllers	To wait for direct recovery state
Performance	Workload, N° of controller running	Minimum controller workload	To wait for direct recovery state

QoS measurements tracks the system operation from run-time information that depends on the type of QoS. For instance, QoS Monitor for Availability must track the liveliness of the controllers, while Load Balancing may monitor current workload of all controllers. Note that if a new controller containing a particular set of MCs is plugged-in and the workload balance QoS is set, the corresponding QoS Monitor should trigger a reconfiguration event that will eventually launch some of the MCs in the new controller, after de-activating them in current controllers.

The *D&D* processes reconfiguration events resolving possible simultaneous triggers and, depending on the problem detected, launches negotiation processes in order to recover the system. Diagnosis is necessary to assure that the system state is known and the QoS can be restored.

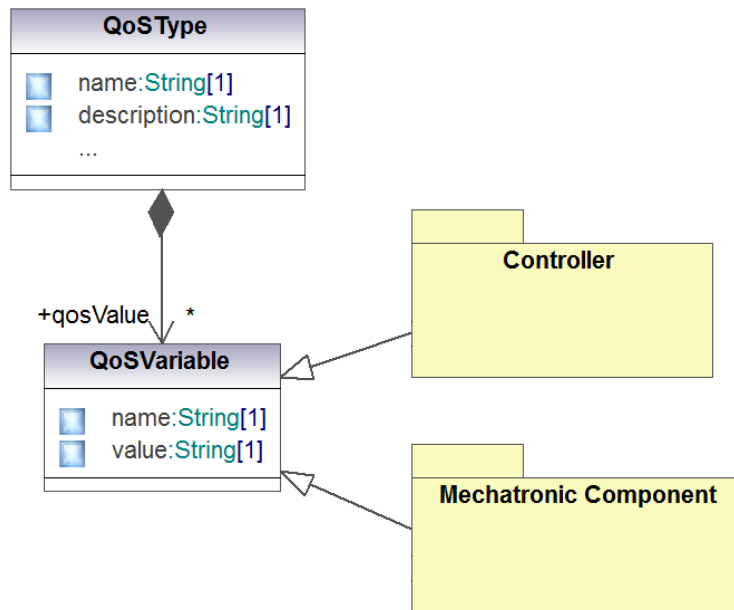
Two different situations are possible: hard QoS loss, meaning that the system is not operating globally, as in the case of controller failure and soft QoS loss, meaning that the operation is not optimal. In the first case, the analysis of the current execution state ( $S_i$ ) of the involved MCs informs about the possible direct launch of MCs in other controllers (direct recovery), the possible launch from a known previous state

(checkpoint recovery) or a safe stop procedure is needed. Critical intervals are defined in  $\Sigma_i$  - *Expression* element of the MC model and recovery actions in the output alphabet and assigned by the output function,  $\delta_i: S_i \times \Sigma_i \rightarrow \Psi_i$ . It is important to note that, in order to be able to perform the diagnosis of the execution state, the D&D needs to have updated information of all MC state.

For the case of soft QoS loss, like in the case of load balancing, the recovery is not time sensitive as the system is operating. Commonly, the decision implies MC redistribution that is carried out at direct recovery states. This reconfiguration prevents the appearance of unpredictable process behaviours.

The information required by the D&D for re-distributing of the MCs, is not limited to the critical interval of the MCs. Depending on the QoS to recover, the re-distribution algorithm may need specific information regarding the MCs and/or the controllers of the system. To illustrate this, let us assume the case of load balancing or energy efficiency. In such cases, it is essential to know the maximum *workload* a MC can introduce into a controller that derives from the MC load in a reference computer and the controller *CPU factor* with respect to a reference CPU. Thus, depending on the QoS type it is necessary to define information needed by the decision algorithm that must be included in the MC or controller models.

This characterization of the controllers and MCs is done using the meta-model presented in Fig. 3-6. In this meta-model, each QoS is defined as a *QoS type* and contains a set of *QoS Variables* that may extend the controller and/or MC models. For instance in the case of load balancing, all MCs will have a *workload* value corresponding to the reference CPU and the controllers are extended with the *CPU factor*.



*Fig. 3-6 QoS characterization meta-model*

## 3.4 Conclusions

This chapter presents a customizable and extensible architecture able to assure the fulfilment of multiple types of QoS of a manufacturing control system. The proposed architecture consists of a series of modules and mechanisms that provide the monitoring, diagnosis, and recovery needed to maintain QoS during the execution of the system.

Different methods for recovering multiple QoS have been presented as part of the architecture. These methods are based on the existence of critical and non-critical execution states, which allows determining when and how a mechatronic component can be recovered. The information required to provide this type of recovery is integrated as part of the complete definition of the manufacturing control system, including the extension of the concept of mechatronic component.

This information has been formalized and included in the control system information model. The definition method for the information model is also presented. This method describes the different steps and requirements that help and guide the

definition of the different elements that compose the control system information model.





## **4 AGENT BASED MIDDLEWARE ARCHITECTURE**

---





## 4.1 Introduction

This chapter presents the definition of a flexible automation middleware (FAM), which implements the runtime platform of the QoS management architecture presented in Chapter 3.

The FAM extends JADE (Java Agent DEvelopment Framework) (Bellifemine et al., 2001; Bellifemine et al., 2008) with a set of agents that allow managing the whole system QoS requirements for manufacturing control applications. JADE is a software framework that aids developers to build agent applications in compliance with FIPA (Foundation for Intelligent Physical Agents) specifications (Intelligent Physical Agents, 2015) for inter-operable intelligent multi-agent systems.

FIPA is based on the assumption that only the external behaviour of system components should be specified, leaving implementation details and internal architectures to platform developers. Based on this assumption, FIPA identifies the roles of some key agents necessary for managing the platform, and describe the agent management content language and ontology. These roles are implemented in the three mandatory agents illustrated at the bottom part of Fig. 4-1:

- The *Agent Management System* (AMS), responsible for agent creation, removal and migration mechanisms.
- The *Agent Communication Channel* (ACC), in charge of interoperability within and among different platforms.
- The *Directory Facilitator* (DF) that supplies a yellow page service to the agent platform where agents can register services or look for required services.

The purpose of JADE is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. To achieve such a goal, JADE offers the following list of features to agent programmers:

- FIPA-compliant Agent Platform, which includes the AMS, the DF, and the ACC
- Distributed agent platform. The agent platform can be split on several hosts; where a Java Virtual Machine represents each host.
- Programming interface to simplify registration of agent services with one or more domains (i.e. DF).
- Light-weight transport mechanism and interface to send/receive messages to/from other agents.
- Library of FIPA interaction protocols ready to be used;
- Automatic registration of agents within the AMS;
- FIPA-compliant naming service: at start-up agents obtain their GUID (Globally Unique Identifier) from the platform.
- Multiple types of Behaviours (SimpleBehaviour, TickerBehaviour, ComplexBehaviour, FSMBehaviour, ...) an agent can use to engage in different simultaneous conversations and activities.

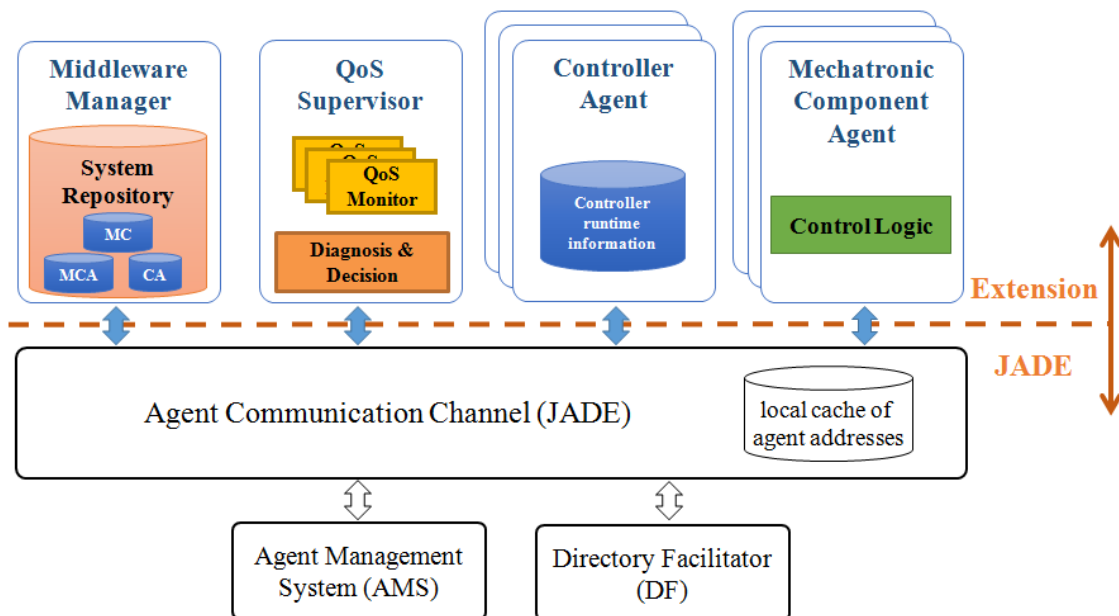


Fig. 4-1 Middleware Implementation

The proposed architecture adds four types of agents (top part of Fig. 4-1). Some of them are part of the basic architecture while there are specific agents for managing specific QoS. Finally, a set of agents that depends on the particular application are in charge of collecting information from the current operation state of the automation system.

- The *Middleware Manager* (MM) agent is unique on the system (although it can be redundant using the services provided by JADE. This extension is out of the scope of this work). It is the main orchestrator and manages the System Repository (SR), containing dynamic information about current state of the automation system (controllers and MCs).
- The *QoS Supervisor* that is composed by *QoS Monitor* (QM) agents, existing as many as QoS to be handled, and one *Diagnosis & Decision* (D&D) agent. Jointly, they supervise QoS fulfilment and launch diagnosis and decision algorithms when needed.

These latter comprise the basic MAS-based architecture for a set of QoS. In addition, there exists a set of agents representing the actual manufacturing application, the so-called application agents:

- There are as many *Controller Agents* (CA) as number of controllers in the system.
- Every MC in the application has associated as many MC agents (MCA) as the number of controllers in the system can run it. At runtime, only one of them, the active MCA, is responsible for controlling the execution of the corresponding MC. It is also in charge of managing MC reconfiguration using the information about the MC formulated in section 3.3.1.

The following sections describe the function of each agent, its role in the architecture and the interaction ontology.

## 4.2 Middleware Manager

The MM is responsible for maintaining the System Repository that contains the current state of the automation system: current active controllers and active MCAs. This is performed through `get_info` and `set_info` ontology commands. This information is updated and read by the supervisor agents (QoS monitors and D&D agents). At system start-up, the model of the automation system is registered: controllers characterized by resources (MCs a CA can run and their corresponding MCAs, memory, CPU factor with respect to the reference controller, etc.) and the set of MCs in the whole automation system. Fig. 4-2 depicts the SR Meta-Model.

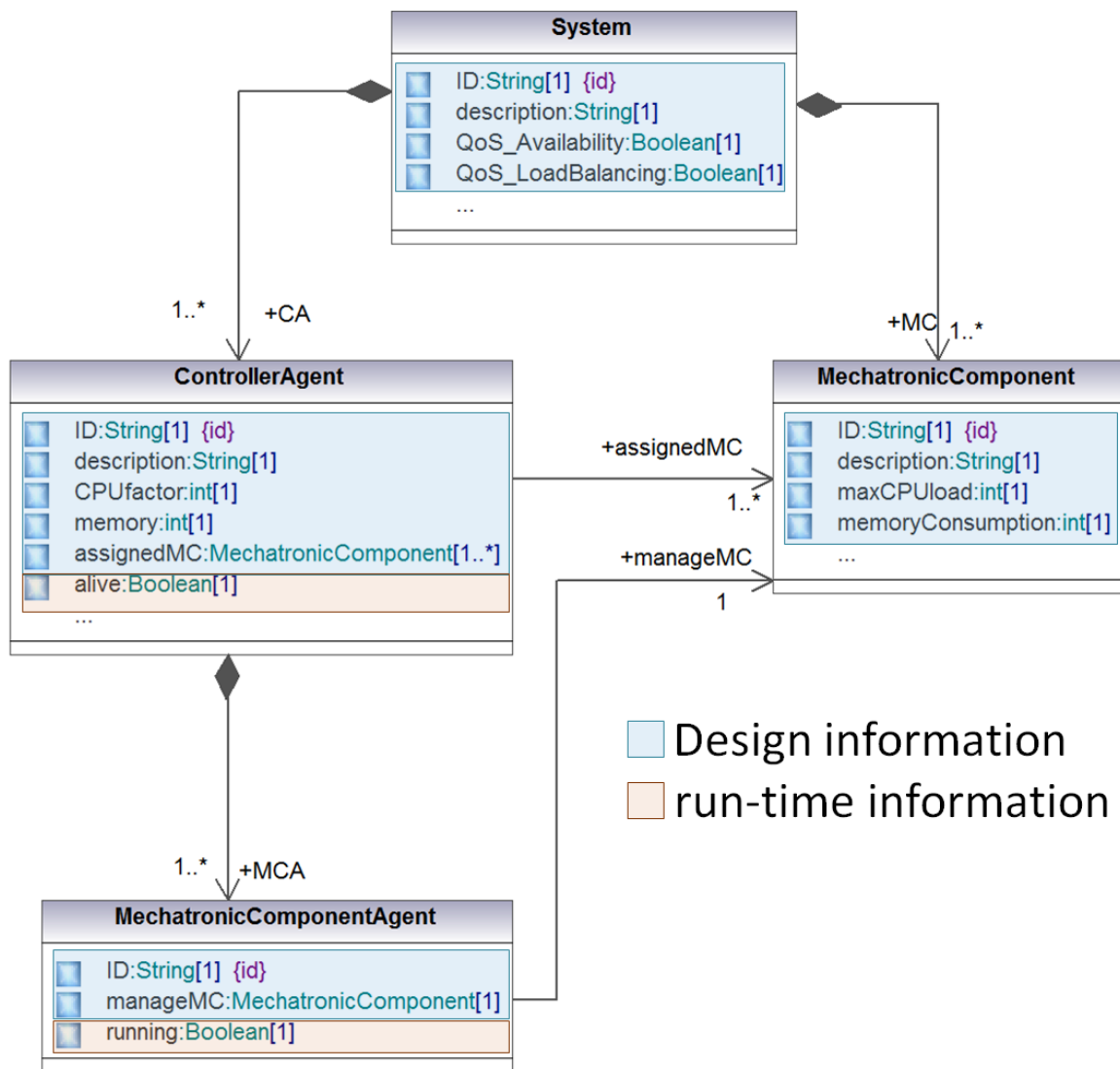


Fig. 4-2 Middleware Manager System Repository

The flexible automation system is characterized by its unique *ID*, a textual *description* of the system and the different QoS to be monitored (QoS can be activated or deactivated at run-time).

As part of the definition of the system, each MC is defined by their:

- *ID*: unique identifier
- *description*.
- *maxCPULoad*: maximum CPU load introduced by the MC in a reference controller.
- *memoryConsumption*: storage space of the control code.

On the other hand, the controllers (CAs) of the system are characterized by:

- *ID*: unique identifier
- *description*.
- *CPUfactor*: with respect to the reference controller.
- *memory*: storage space provided by the controller
- *AssignedMC*: ids of the MCs that can be executed in the controller.
- *alive*: activity variable that defines whether the controller is active or not.

Finally, the CAs have a MCA for each of the MCs assigned to them. These MCAs are defined by:

- *ID*: unique identifier

- *manageMC*: id of the MC whose execution is managed by the agent
- *running*: activity variable that informs whether or not the execution of MC is being performed by this agent.

To provide access to the SR, at boot time the MM registers in the DF the service “SystemRepository”.

## 4.3 QoS Supervisor

The supervisor is responsible for tracking a set of system QoS, detecting QoS losses and making decisions in order to recover the lost QoS as soon as possible. Its functionality is performed by a set of QM agents and one D&D agent.

### 4.3.1 QoS Monitor (QM) Agents

Each QM is responsible for monitoring specific QoS meeting and registers itself in the DF as a monitoring service (e.g. *Monitoring\_Availability* or *Monitoring\_SystemEfficiency*).

The monitoring is performed by means of the collaboration between QM and application agents (MCAs and CAs). Application agents are in charge of detecting situations that can lead to QoS loss (for instance, loss of controller heartbeat in Availability or maximum workload overtaken in case of system efficiency) and inform to the corresponding QM accordingly. Upon the reception of QoS loss events, the QM sends confirmation requests to avoid false positives, registers the type of QoS lost to handle subsequent events of the same type and launches reconfiguration events.

### 4.3.2 Diagnosis & Decision (D&D) Agent

The D&D is responsible for recovering QoS if it is possible and registers itself in the DF as such (“DiagnosisDecision”). It receives reconfiguration events from QMs. As

multiple reconfiguration events corresponding to different QoS can be issued, D&D analyses the global system state and makes decision to recover QoS by priority. Each QoS requires specific analysis but reconfiguration actions always involved the activation of MCs in other controllers, sometimes after de-activation of MCs and / or execution of recovery actions, depending on the situation.

To perform analysis processes, the D&D involves application agents (CAs and MCAs) while to make a decision it launches negotiation processes to get the CA in which a MC will be activated. Finally, it uses diagnosis processes performed by MCAs to launch the reconfiguration or stop the MC, warning the operator in case of severe errors.

## **4.4 Application Agents**

The application agents are in charge of monitoring resources and managing the execution of the MCs. There are as many AS, as controller in the system and as many MCAs per MC as controllers are able to run the MC.

### **4.4.1 Controller Agent**

When a controller joins the system, the corresponding Controller Agent (CA) is launched and it registers the controller and associated resources in the SR. It also registers itself in the DF offering as services the set of MCs that can run in the controller. Finally, it launches an MCA for each MC.

As part of its functionality, CAs may perform QoS monitoring functions. For instance, for system efficiency QoS, the CA can monitor the current workload of the controller, issuing QoS loss when the workload is less than the minimum or greater than the maximum allowed. It may also be involved in negotiation processes launched by D&D.

## 4.4.2 Mechatronic Component Implementation

Since not all commercial IEC 61131-3 execution environments allow changing the source code of the controller at run-time, the solution proposed is to deploy in the controllers the set of MCs they can eventually run. The MCA controls the execution of the PLC code and the run-time platform. It assures that at any time only one of the MC replicas is running. To do that the MC code is instrumented.

Thus, a MC is implemented as an agent, the MCA, running under JADE and a piece of instrumented code running in the PLC virtual machine. The following sub-sections describes both, the agent functionality and the MC code instrumentation.

### 4.4.2.1 Mechatronic Component Agent

As commented above, MCAs are in charge of managing the execution of the corresponding MC control code. Each MCA is able to activate / de-activate the MC, launch the execution of recovery actions as well as collect, transmit, store and make diagnosis on the current state of the MC. MCA operation is defined by the Finite State Machine (FSM) depicted in Fig. 4-3. States correspond to JADE agent behaviours and transitions between states are triggered by the D&D as a result of a decision.

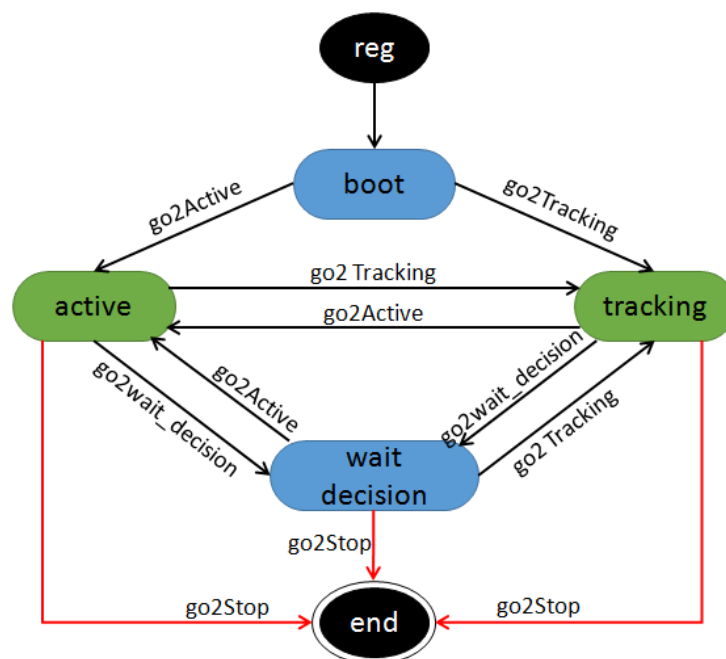


Fig. 4-3 MCA Finite State Machine (FSM)



- *Boot* state: It corresponds to the start-up where initialization actions are performed as well as self-registration in the SR.
- *Active* state: While in this state, the MC is executed in the PLC runtime and MCA collects periodically the MC current state accessing to the PLC memory and transmits it to the rest of MCAs that are tracking the active one. Note that only one MCA of the set corresponding to each MC can be in the active state.
- *Tracking* state: While in this state, the MCA receives and stores the current state of the MC, sent by the active MCA.
- *Wait decision* state: During the reconfiguration process the MCA remains in this state attending requests from the D&D, such as MC state diagnosis or stop MC execution in the next noncritical state (in order to perform a direct MC reconfiguration from such state). To perform MC state diagnosis, MCAs make use of a set of masks derived from the critical interval expressions (see Section 3.3.1).
- *End* state: When the MCA enters this state, the agent is removed after clean-up tasks are executed.

Note that the functionality executed in every state may be extended to handle different type of QoS. For instance, while in the tracking state, MCAs can monitor a problem in service availability if the current MC state is not received within a timeout. As part of the functionality of this state, the MCA can issue a QoS loss event due to a possible failure in the active MCA.

#### 4.4.2.2 MC code

In order to perform *MC code instrumentation*, source code insertion (SCI) technology is used. It is necessary to integrate three different programs:

1. A serialization program that collects the value of the variables that compose the state of the MC, allowing the MCA to access the current execution state.
2. A de-serialization program that extracts the values of the state variables from the state provided by the MCA and writes them into the PLC memory.
3. An execution manager program in charge of receiving orders from the MCA to activate/deactivate the execution of the MC PLC program as well as executing recovery and stop actions. This functionality is achieved through management variables written by the MCA and read by the MC code. Fig. 4-4 illustrates this program that distinguishes three different situations: MC in execution, MC activation, with possibly recovery actions or safe stop and MC de-activation.

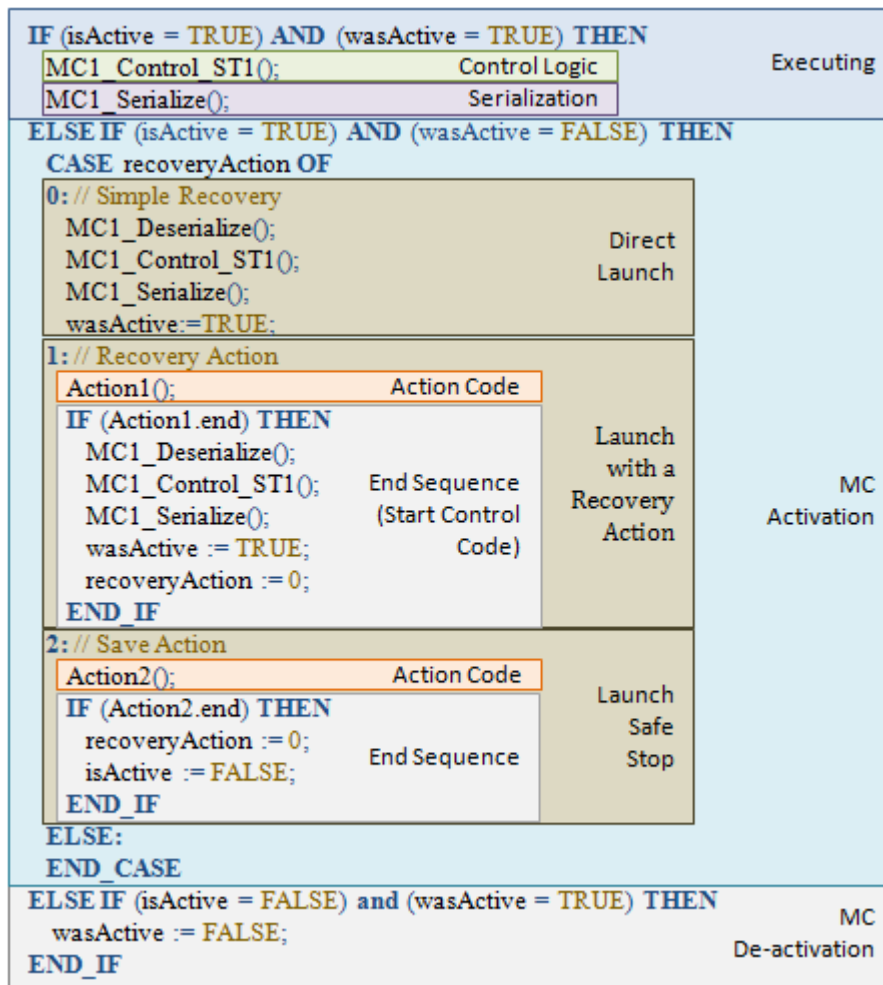


Fig. 4-4 Execution control code template

- **Executing** (*isActive*=TRUE and *wasActive*=TRUE): this situation corresponds to the MC execution. Thus, the control logic of the MC is executed (*MCI\_Control\_ST1*) followed by the serialization of the state program (*MCI\_Serialize*), enabling the state reading from the MCA.
- **MC Activation** (*isActive*=TRUE and *wasActive*=FALSE): This situation occurs when the MC in this controller was inactive and receives an activation order. The MCA indicates if it is a direct activation, activation after executing a recovery action or a safe stop action. There exists three different code sequences depending on the type of action required:
  - **Direct Launch**: It happens when the MC can execute from the current state (*recoveryAction*=0). This situation is also valid for the case of checkpoint recovery, as the MCA will have written the checkpoint state. Thus, after state de-serialization, the MC control logic executes and the new state is serialized. Finally, *wasActive* is set to TRUE in order to indicate to the next cycle that the MC was active in the previous one.
  - **Launch with Recovery Action**: this is used when a recovery action must execute first. The execution state is only de-serialized when the recovery action is finished; then, the MC control logic executes and the new state is serialized. Finally, for preparing the next cycle, *wasActive* is set to TRUE.
  - **Launch Safe Stop**: This situation means that a failure has occurred during a critical interval and the MC cannot be recovered. Thus, the corresponding stop action is performed. After finishing this execution, the MC is de-activated by setting *isActive* to FALSE.
- **MC De-activation** (*isActive*=FALSE): this situation means that the MC in this controller is not in execution. It will remain in this situation until a new order from the MCA is received.

## 4.5 QoS Management Ontology

Agent communication is based on message passing, where agents communicate by sending individual messages to each other. Each message needs to follow a specific syntax and semantics enabling agent interaction. The concepts, syntax and semantics are known as message ontology.

The particular format of JADE messages is fully compliant with FIPA Agent Communication Language (ACL). Each message includes the following fields:

- The **sender** of the message.
- The list of **receivers**.
- The communicative act (also called the '**performative**') indicating what the sender intends to achieve by sending the message. For instance, if the performative is REQUEST, the sender wants the receiver to perform an action, if it is INFORM the sender wants the receiver to be aware of a fact, if it is a PROPOSE or a CFP (Call for Proposals), the sender wants the receiver to enter into a negotiation.
- The **content** containing the actual information to be exchanged by the message (e.g., the action to be performed in a REQUEST message, or the fact that the sender wants to disclose in an INFORM message, etc.).
- The **ontology** identifying the format and meaning of the content. Both, the sender and the receiver must understand the same meaning for the communication to be effective.
- The **conversation-id** which presents the unique identifier of a conversation thread.

- Some additional fields used to control several concurrent conversations and to specify timeouts for receiving a reply such as *reply-with*, *in-reply-to* and *reply-by*.

The characterization of these fields provides the definition of the message ontology used during agent communication.

Fig. 4-5 illustrates the different messages used by the middleware and the agent that generates them.

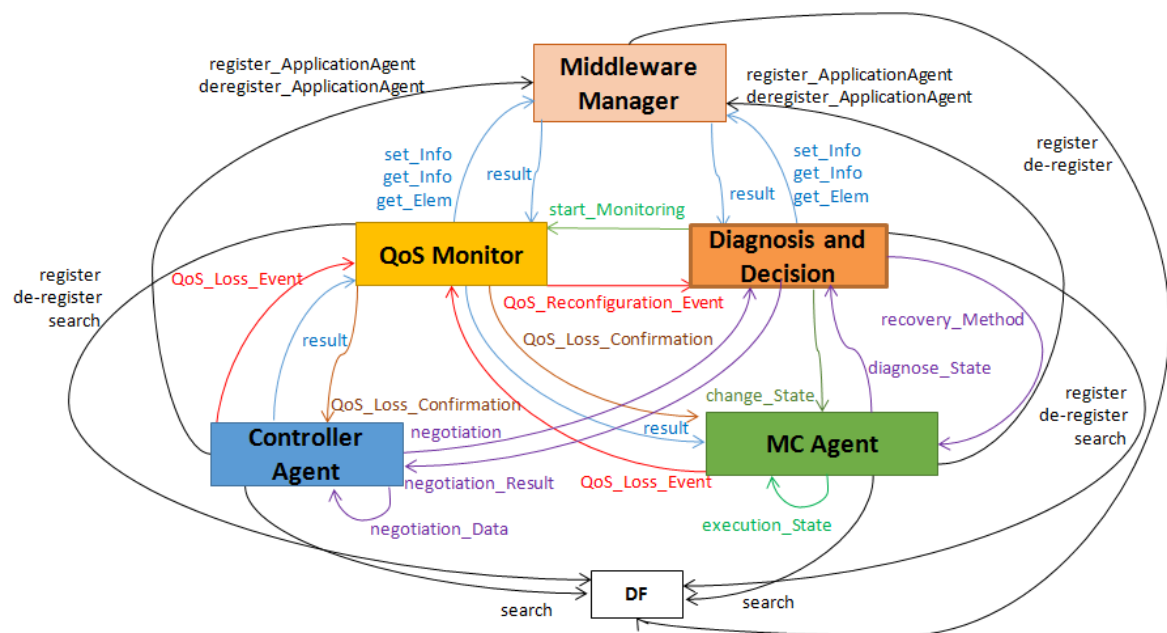


Fig. 4-5 Middleware Message Ontology

The following subsections present the syntax and semantics of the different messages that form the ontology.

#### 4.5.1 Registration messages

These messages allow registering and de-registering of the application agents in the SR and the middleware services in the DF.

<b>register_ApplicationAgent:</b> message used to register a CAs and MCAs in the SR	
<b>performative</b>	Request
<b>content</b>	<pre>register_ApplicationAgent(agentType; {parameter=value})</pre> <ul style="list-style-type: none"> <li>- agentType: type of agent to be register</li> <li>- {parameter=value}: list of names and values of the parameters to of the new element</li> </ul>
<b>example</b>	<pre>register_ApplicationAgent(CA;ID=CA1;description=controller1;ip=192.168.1.101;CPUfactor=1;memory=250;MC=MC1,MC2,MC3) register_ApplicationAgent(MCA;ID=MCA1;parentCA=CA1;manageMC=MC1)</pre>

<b>deregister_ApplicationAgent:</b> message used to de-register a CAs and MCAs from the SR	
<b>performative</b>	Request
<b>content</b>	<pre>deregister_ApplicationAgent(agentID)</pre> <ul style="list-style-type: none"> <li>- agentID: Identifier of the agent to de-register</li> </ul>
<b>example</b>	<pre>deregister_ApplicationAgent(CA1) deregister_ApplicationAgent(MCA1)</pre>

<b>register:</b> message used to register a service in the DF	
<b>performative</b>	Request
<b>content</b>	<pre>register(service;agentID)</pre> <ul style="list-style-type: none"> <li>- service: name of the service to be register</li> <li>- agentID: identifier of the agent that provides the service</li> </ul>
<b>example</b>	<pre>register(SystemReository;MM) register(DiagnosisDecision; DD) register(QoS_Availability;QM1) register(QoS_SystemLoad;QM2)</pre>

<b><i>de-register</i></b> : message used to de-register the services associated to an agent	
<b>performative</b>	Request
<b>content</b>	de-register(agentID) - agentID: identifier of the agent to de-register
<b>example</b>	de-register(MM) de-register(DD) de-register(QM1) de-register(QM2)

## 4.5.2 Information messages

These messages are used by the different agents to access and modified the information contained in the SR and the DF.

<b><i>set_Info</i></b> : message used to request the modification of a parameters in a specific element of SR.	
<b>performative</b>	Request
<b>content</b>	set_Info(elementID; parameter=value) - elementID: Identifier of the element to be modified - parameter=value: name and value of the parameter to be modified
<b>example</b>	set_Info(CA1; alive=false) set_Info(MCA1; running=true)

<b><i>get_Info</i></b> : message used to request the value of a certain parameter of an element.	
<b>performative</b>	Request
<b>content</b>	get_Info(elementID; parameter) - elementID: Identifier of the element to access - parameter: name of the parameter that has been requested
<b>reply-with</b>	result message
<b>conversation id</b>	Generate

<b>example</b>	get_Info(CA1; CPUfactor) get_Info(MCA1; manageMC)
----------------	------------------------------------------------------

<b>get_Elem</b> : message used to request the elements whose parameters match the ones specified in the message.	
<b>performative</b>	Request
<b>content</b>	get_Elem(elementType; {parameter=value}) <ul style="list-style-type: none"> <li>- elementType: type of elements to be compared</li> <li>- {parameter=value}: list of names and values of the parameters to use in the comparison.</li> </ul>
<b>reply-with</b>	result message
<b>conversation id</b>	Generate
<b>example</b>	get_Elem(MCA; parentCA=CA1;running=true) get_Elem(CA; MC=MC1)

<b>result</b> : reply message containing the value of required information	
<b>performative</b>	Inform
<b>content</b>	result(data) <ul style="list-style-type: none"> <li>- data: reply information</li> </ul>
<b>conversation id</b>	Reference
<b>example</b>	result(MCA1) result(MCA2,MCA3)

<b>search</b> : search for the agent associated to a service or the service associated to an agent	
<b>performative</b>	inform
<b>content</b>	search(searchParameter) <ul style="list-style-type: none"> <li>- searchParameter: name of the agent or service to be located</li> </ul>



<b>example</b>	<pre>search(MiddlewareManager) search(DiagnosisDecision) search(QoS_Availability) search(QoS_LoadBalancing)</pre>
----------------	-------------------------------------------------------------------------------------------------------------------

### 4.5.3 QoS loss messages

Group of messages used during the detection of a QoS loss, the verification and launch of the reconfiguration.

<b>QoS_Loss_Event:</b> message used by the application agents to inform of a situation that can lead to QoS loss.	
<b>performative</b>	inform
<b>content</b>	<pre>QoS_Loss_Event(situation; elementID) - situation: type of situation - elementID: identifier of the element involved in this situation</pre>
<b>conversation id</b>	generate
<b>example</b>	<pre>QoS_Loss_Event(MC_failure; MC1) QoS_Loss_Event(upperLimit_reach; CA1) QoS_Loss_Event(lowerLimit_reach; CA1)</pre>

<b>QoS_Loss_Confirmation:</b> request issued by QM to confirm the situation of a application agent. The reply to this request is done using a result message.	
<b>performative</b>	request
<b>content</b>	<pre>QoS_Loss_Confirmation(situation) - situation: type of situation to verify</pre>
<b>reply-with</b>	result message
<b>conversation id</b>	generate
<b>example</b>	<pre>QoS_Loss_Confirmation(is_active) QoS_Loss_Confirmation(upperLimit_reach)</pre>

	QoS_Loss_Confirmation(lowerLimit_reach)
--	-----------------------------------------

<b>QoS_Reconfiguration_Event:</b> message use by QM to trigger the reconfiguration.	
<b>performative</b>	inform
<b>content</b>	QoS_Reconfiguration_Event (QoSType, reconfigurationType, agentID) <ul style="list-style-type: none"> <li>- QoSType: QoS that has failed</li> <li>- reconfigurationType: type of reconfiguration to be performed</li> <li>- agentID: agent to be reconfigured</li> </ul>
<b>reply-with</b>	start_Monitoring message
<b>conversation id</b>	generate
<b>example</b>	QoS_Reconfiguration_Event(availability, MCA_recovery, MCA1), QoS_Reconfiguration_Event(availability, CA_recovery, CA1) QoS_Reconfiguration_Event(systemLoad, upperLimit_recovery, CA1) QoS_Reconfiguration_Event(systemLoad, lowerLimit_recovery, CA1)

#### 4.5.4 Negotiation messages

These messages are used during CAs negotiation processes.

<b>negotiation:</b> message used by D&D to start a negotiation between different CAs.	
<b>performative</b>	request
<b>content</b>	negotiation(negotiationCriteria; participants) <ul style="list-style-type: none"> <li>- negotiationCriteria: define the winning criteria</li> <li>- participants: list of CAs involved in the negotiation</li> </ul>
<b>reply-with</b>	negotiation_Result message
<b>conversation id</b>	generate
<b>example</b>	negotiation(minimum workload; CA2,CA3)

<b><i>negotiation_Data</i></b> : message used by CAs to exchange information during the negotiation process.	
<b>performative</b>	inform
<b>content</b>	negotiation_Data(data) - data: information to be exchanged
<b>conversation id</b>	generate
<b>example</b>	negotiation_Data(15.64)

<b><i>negotiation_Result</i></b> : message use by winner CA to inform the D&D of the result of the negotiation.	
<b>performative</b>	inform
<b>content</b>	negotiation_Result(data) - data: information to be exchanged
<b>conversation id</b>	reference
<b>example</b>	negotiation_Result(15.64)

### 4.5.5 Diagnosis messages

Message used during the diagnosis of the execution state.

<b><i>diagnose_State</i></b> : message used by the D&D to request the diagnosis of the execution state	
<b>performative</b>	request
<b>reply-with</b>	recovery_Method message
<b>conversation id</b>	generate
<b>example</b>	diagnose_State()

<b>recovery_Method</b> : message use by MCA to inform the D&D of the result of the diagnosis.	
<b>performative</b>	inform
<b>content</b>	recovery_Method(recoveryType) <ul style="list-style-type: none"> <li>- recoveryType: type of recovery associated to the current execution state.</li> </ul>
<b>conversation id</b>	reference
<b>example</b>	recovery_Method(direct) recovery_Method(checkpoint) recovery_Method(non-recovery)

### 4.5.6 Reconfiguration messages

These messages are related to the action used by the D&D to recover the QoS and the normal execution of the system.

<b>change_State</b> : message used by D&D to request a state change to a MCA. This message can contain additional information related to actions to be performed at state entrance. For example, perform a direct recovery, wait for a noncritical state, among others.	
<b>performative</b>	request
<b>content</b>	change_State(FSMstate; stateAction) <ul style="list-style-type: none"> <li>- FSMstate: name of the target state</li> <li>- stateAction: action to be performed before entering the state</li> </ul>
<b>conversation id</b>	reference
<b>example</b>	change_state(active) change_state(active, direct) change_state(active, checkpoint) change_state(tracking) change_state(waitDecision) change_state(waitDecision, nonCriticalStop) change_state(stop) change_state(stop, nonRecovery)

<b>start_Monitoring:</b> message use by D&D to inform the QM of the end of a reconfiguration process	
<b>performative</b>	request
<b>conversation id</b>	reference
<b>example</b>	start_Monitoring()

### 4.5.7 State message

This message syntax is used by the active MCAs to send the state to the tracking MCAs.

<b>execution_State:</b> message use by active MCAs to send the state to the tracking MCAs	
<b>performative</b>	inform
<b>content</b>	execution_State(state) - state: current value of the execution state.
<b>example</b>	execution_State([0,0,1,0,0,0,0,...])

## 4.6 Assuring QoS

This section presents the customization middleware agents and the use of message ontology to ensure the fulfilment of system QoS based on QOS monitoring, loss detection and recovery. To test out this middleware two different QoS have been selected:

- **Service availability:** which is able to detect service interruptions and analyze whether or not service can recover.
- **System Efficiency:** in the sense of ensuring that, all controllers work with the smallest time cycle as possible.

These two types of QoS are representative enough as to illustrate the flexibility the middleware can provide. The first provides a recovery based on the last known state

of the execution, while the latter focuses on a reconfiguration during non-critical states.

### 4.6.1 Availability QoS

Availability implies the service continuity or service recovery as soon as possible and transparently to the application. The following sub-sections illustrate the middleware operation in case of controller failure from availability monitoring, loss detection under a controller failure up to recovering the service on other controllers. Customized sequence diagrams (to include timeouts) are used to show the messages exchange and the agents involved in each phase.

#### 4.6.1.1 QoS monitoring

As commented above, depending on the QoS to be handled, there are application agents (CAs, MCAs) responsible for monitoring QoS meeting and issuing QoS loss events to the QM. Thus, the agent code has been extended to handle such functionality. In the case of service availability, MCAs in tracking state are responsible for availability monitoring. They receive the current state of the MC from the active MCA. The reception of the state represents the active MCA heartbeat. If tracking MCAs do not receive the state within a timeout, they issue Availability QoS loss to QM through *QoS\_Loss\_Event* message. The message contains the situation detected (in this case MC service unavailability) and the affected resource (MC1). Fig. 4-6 illustrates the functionality extension of the tracking state.

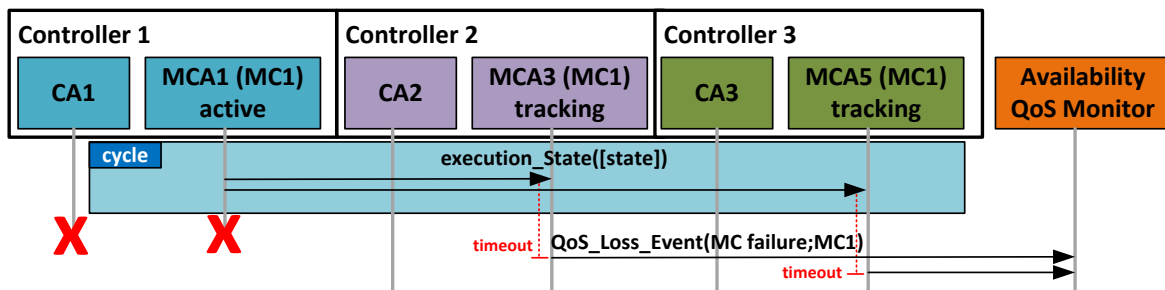
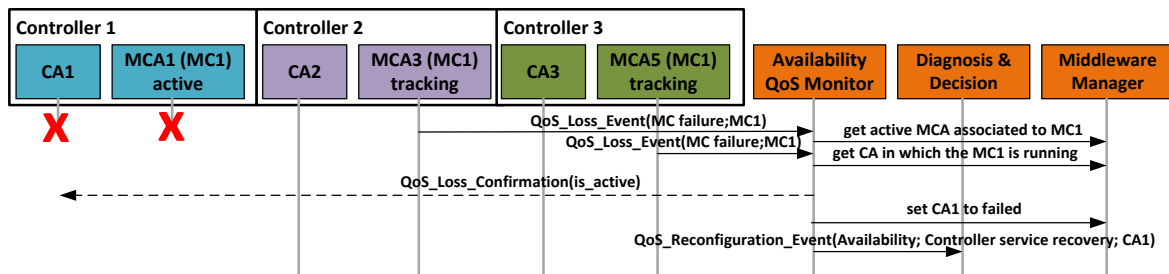


Fig. 4-6 Availability QoS Monitoring and QoS loss detection

### 4.6.1.2 QoS loss detection

Upon the reception of the Availability loss event, QM registers MCA failure to avoid multiple reactions to the same event. Then QM confirms the QoS loss issuing a confirmation message to the CA. In case a controller failure is confirmed, the failure is registered to avoid reacting to several events triggered by other tracking MCAs or other MCs in failure. Next, a reconfiguration event is issued to the D&D through the *QoS\_Reconfiguration\_Event*, characterized by the QoS affected (availability), type of reconfiguration (controller service recovery), and the agent involved (CA1). Fig. 4-7 illustrates this situation.



**Fig. 4-7 Availability loss detection**

Note that if a controller failure is not confirmed the QM request the state to the active MCA in order to confirm MC failure or a false positive.

### 4.6.1.3 Diagnosis, Decision and Recovery Actions

The D&D resolves concurrent reconfiguration events corresponding to different QoS by priority (for instance service availability has higher priority than system efficiency). If it corresponds to a controller failure, D&D gets from the SR the MCs to recover and issues messages to all their MCAs to provoke a state transition to the *wait\_decision* state. Next, it proceeds to sequentially recover each of the failed MCs (active MC in the failed controller). The recovery process starts by launching a negotiation process in all CAs containing MC. Once the negotiation finishes, D&D requires the diagnosis of MCs state to the winner MCA. After the recovery method for the MC is known, the D&D proceeds to recover (direct /checkpoint) or stop (non-recovery) the execution of the MC. When all failed MCs have been recovered, the D&D

informs the QM about the finalization of the recovery process (*start\_monitoring*). This is illustrated in Fig. 4-8.

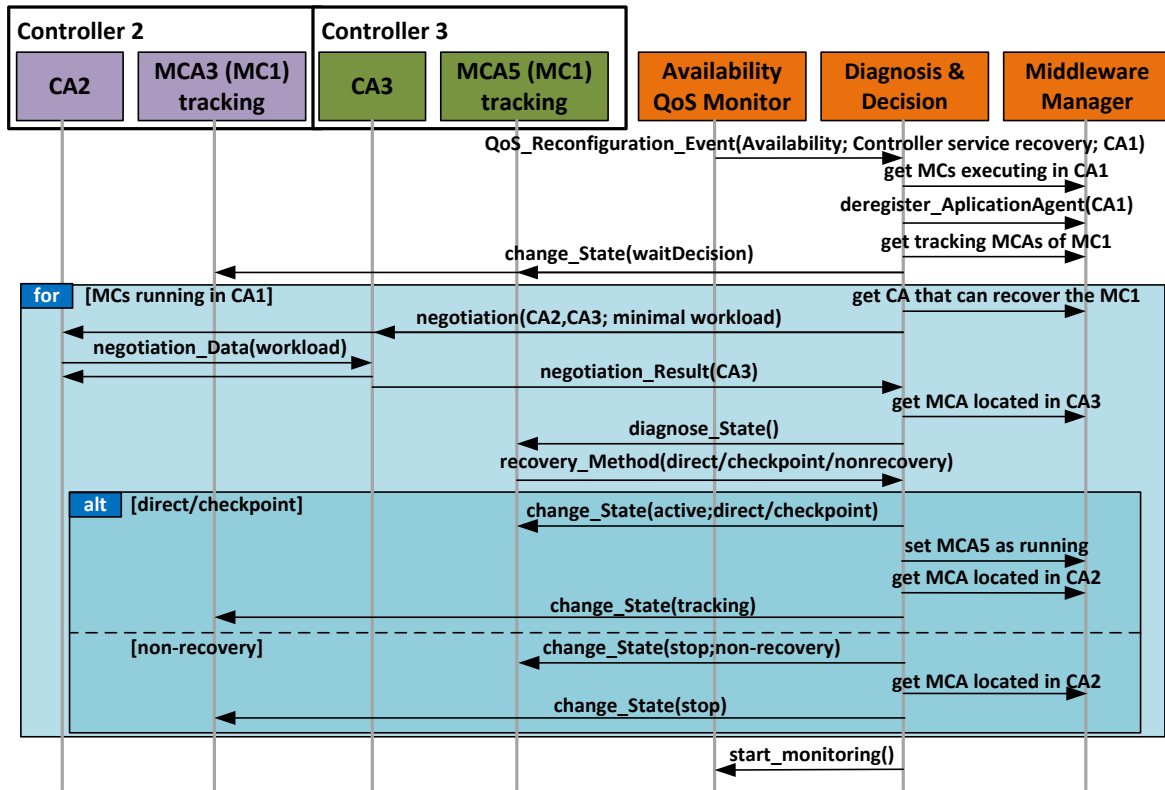


Fig. 4-8 Service recovery through negotiation phase

### 4.6.2 System Efficiency QoS

To illustrate a soft reconfiguration, those characterized by a non-critical situation and not affected by an urgent action, this sub-section presents how assuring system efficiency.

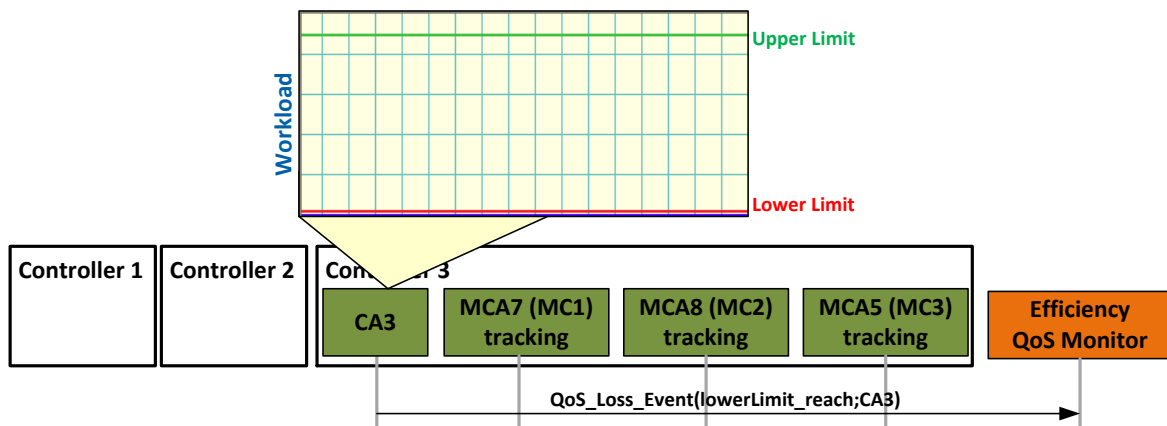
Let us assume a definition of system efficiency as to try to get the best execution cycle for all controllers. In such a case, reconfiguration is launched when the execution cycle of controllers are very different and the goal of the reconfiguration decision is to redistribute the MCs among the different controllers to obtain an optimized workload distribution. In this case, as the management platform is pro-active, the reconfiguration will always be performed during a no-critical state.



The following sub-sections illustrate the middleware operation in case of new controller arrival to the system (controller plug-in), from the QoS monitoring, loss detection under a controller addition up to recovering the workload balance of all controllers.

#### 4.6.2.1 QoS monitoring

As commented above, in the case of system efficiency, the CAs monitors the current workload of their corresponding controllers, issuing a QoS loss event when the workload is less than the minimum or greater than the maximum workload allowed. The QoS loss event message provides information regarding the situation detected (in this case the workload is lower than the limit) and the affected resource (CA3).



**Fig. 4-9 System Efficiency QoS Monitoring and QoS loss detection**

#### 4.6.2.2 QoS loss detection

Upon the reception of the QoS loss event, the QM proceeds to verify the situation by issuing a confirmation message to the CA. Upon the reception of this request, the CA verifies if the workload is lower than the defined limit and informs of the result. In case of confirmation, a QoS reconfiguration event is issued to the D&D. This event is characterized by the affected QoS (system load), the type of reconfiguration (lower limit recovery) and the agent involved (CA3). Fig. 4-10 illustrates this situation.

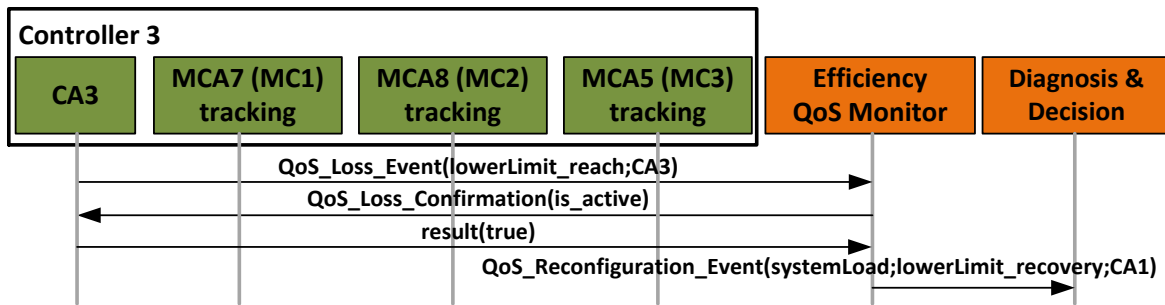


Fig. 4-10 System efficiency loss detection

### 4.6.2.3 Diagnosis, Decision and Recovery Actions

The D&D proceeds to resolve the reconfiguration event. It takes the CPU factors of the CAs, current distribution of the MCs, as well as the maximum CPU load introduced by the MCs (maxCPUload) from the SR and proceeds to balance the workload of the controllers. After obtaining the new distribution, the D&D starts relocating the execution of the MCs. The relocation process begins by issuing transition messages to the MCAs of the MC to provoke a state change to the *wait\_decision* state. During this state the running MCA will stop the execution of the control code in a noncritical state and sends the current state. Once the execution stops, the D&D starts the execution in the new location. This is illustrated in Fig. 4-11.

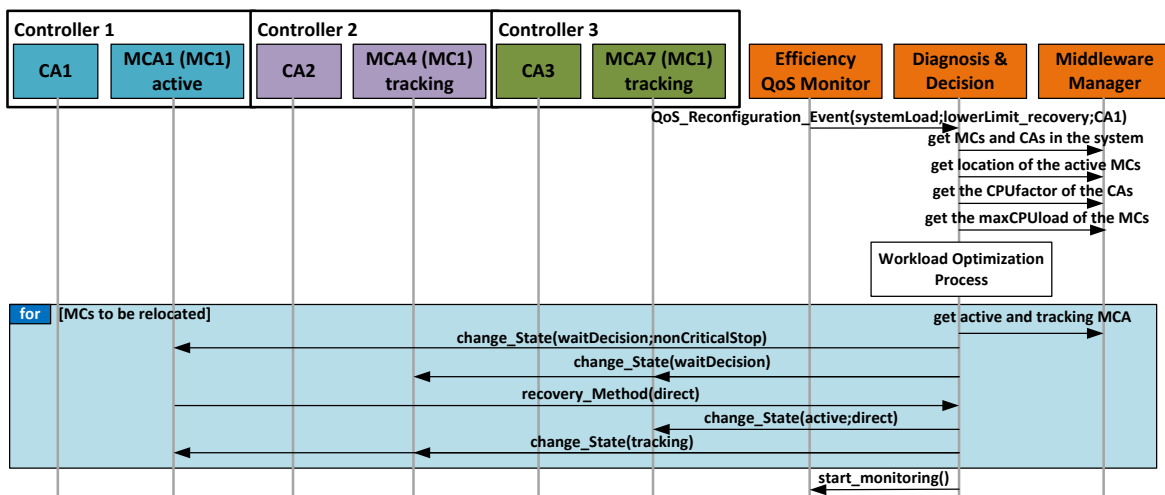


Fig. 4-11 Relocation Service

## 4.7 Conclusions

This chapter focuses on the implementation of the QoS management architecture using an agent based middleware. The basic middleware offers a set of agents to monitor the QoS, resolve multiple QoS loss events and recover the QoS if it is possible.

Guidelines to include other QoS include defining monitoring functions using application agents as well as recovery actions. Besides, information exchange through ontology commands has to be also defined. If necessary, the ontology could be extended with new commands. Thus, the architecture can be instanced to include different monitoring and diagnosis functions by extending the agent's functionality.

As an example of the extension of the QoS, two different test cases have been presented, which exemplify the extension of the application and supervisor agents, as well as the different reconfiguration mechanism the middleware can provide.

However, this implementation presents some drawbacks, which need to be highlighted:

1. As previously commented the controller must contain the code related to each of the MCs associated to it; since most of the commercial controllers do not allow a dynamic download of the control code.
2. The use of a multi-agent system adds to the controller an overhead related to the execution of the CA and MCAs, as well as their communications.
3. The architecture can only run in IEC61131 commercial controllers that provide an operating system in which the JADE can be executed.







## **5 THE FLEXIBLE AUTOMATION FRAMEWORK**

---



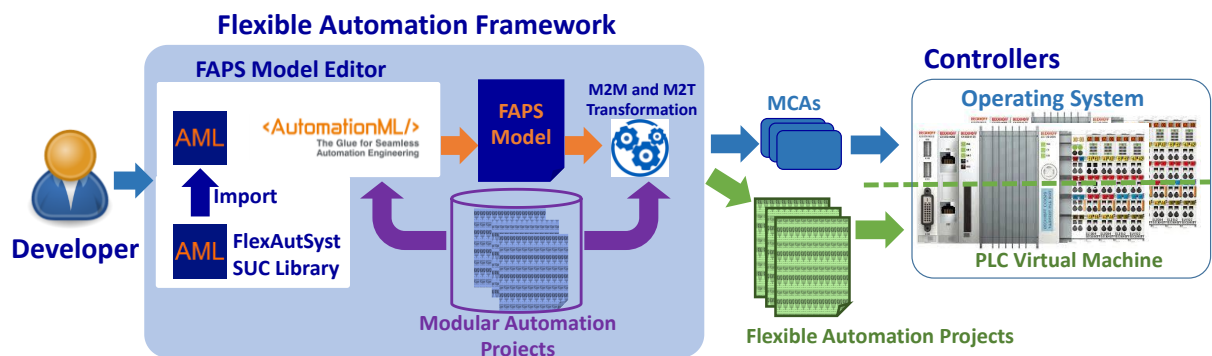


## 5.1 Introduction

This chapter presents a framework aiming at facilitating the developer tasks. The goal of the framework is to automatically generate the so-called flexible automation projects. These are the automation projects to deploy that contain the set of MCs the corresponding controller can run, including the activation / de-activation mechanisms as well as the recovery actions to start from checkpoint states or to perform a safe stop. Additionally, the framework also customizes the templates corresponding to the appropriate MCAs and CAs.

To achieve this goal the framework requires different type of information: the characterization of the controller running both, the PLC virtual machine and the Java virtual machine, the original automation projects containing the control of the process organized in MCs and the information related to the critical intervals of each MC.

The proposed framework, illustrated in Fig. 5-1, makes use of the models of Chapter 3 and uses model transformation techniques to achieve its goal. It is based on automation standards. In particular, PLCopen (Van der Wal, 2009) is used for defining the MCs and Automation ML (Lüder et al., 2011; Anon, 2016) is used to define the system information in terms of controller characterization, MC identification within PLCopen files (code and variables) and the corresponding critical intervals.



**Fig. 5-1 General Scenario of Flexible Automation Framework**

The framework is composed of two main modules: The Flexible Automation Production System (FAPS) Model editor and the code generator. There are two types of generated code; on the one hand the so-called Flexible Automation Projects that contain the instrumented MCs each PLC can run, the activation / de-activation code and the recovery / safe stop actions. On the other hand, it also generates the code corresponding to the application agents: MCAs and CAs.

Although the MCs are one of the inputs to the framework, in order to facilitate the tasks of defining the automation system modularly, an automatic generation tool has been designed, based on the controller information model presented in Chapter 3. This tool is presented in section 5.2.

The rest of the chapter details the FAPS model editor as well as the automatic generation of the Flexible Automation Projects and application agents using model to model (M2M) and model to text (M2T) transformations. Due to the FAPS model is expressed in XML, XML stylesheet technology has been selected to perform M2M and M2T transformations (Schmidt, 2006).

## **5.2 Generation of tool-independent automation projects**

The developer is required to design and develop the automation system modularly, i.e. as a set of MCs (they control part of the process, use a set of input and outputs and their control logic is as a set of POU's and variables). The POU's of the control logic are developed in a PLC programming tool and exported in PLCopen format, if the programming tool supports some type of exportation function, such as the PLCopen interface. In order to guide the definition of the modular automation project, a UML-based modeling tool has been developed that directly generates the PLCopen automation projects.

The UML modeling tool allows the definition of the automation control system from a functional, hardware and software point of view (Control Engineering Domain View,

Electrical-Electronic Engineering Domain View and Software Engineering Domain View). Each view is characterized by its lexicon, defined as a set of three UML profiles and UML class diagrams. The UML profiles are imported into the UML modelling tool. Fig. 5-2 presents the syntax (lexicon and relationships) for each of the views.

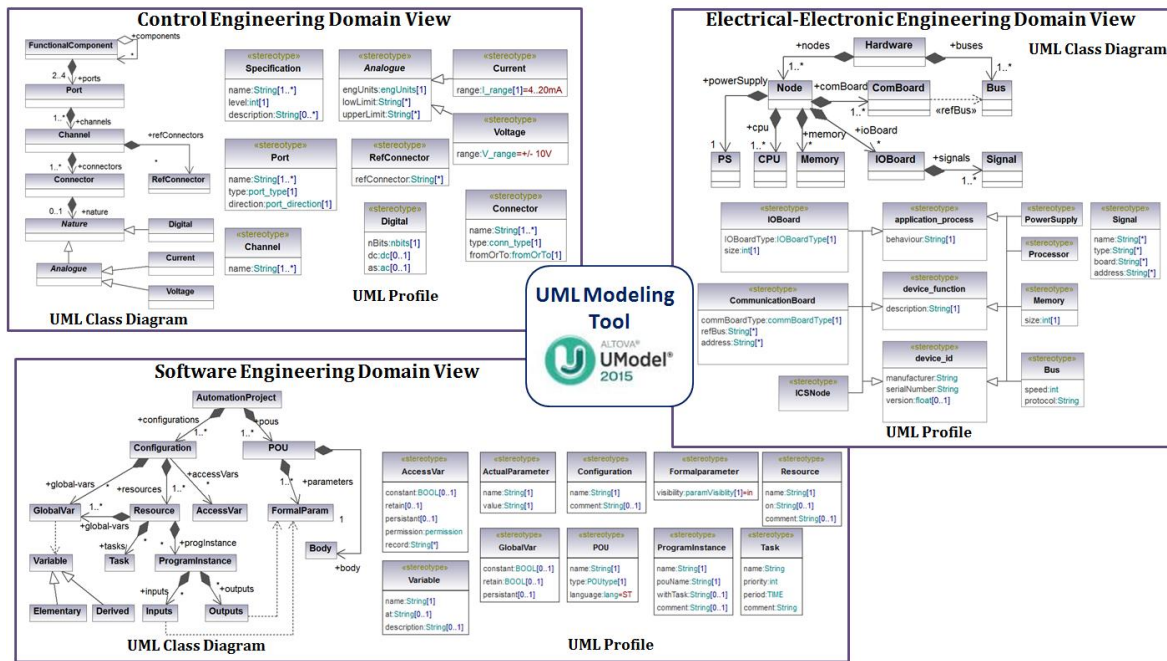


Fig. 5-2 UML class diagrams and UML profiles for the different domain views

The definition of the automation project relies on a series of previously defined POUs. These POUs are programmed and tested within the PLC programming tool (Twincat 3.0, Phoenix Contact, CODESYS 3.5, among others). Once they are validated, the POUs are exported and stored in a repository in PLCopen XML format (step 1).

To integrate the POUs used by the automation project under definition in the UML modeling tool, it is necessary to perform a model transformation. This transformation has as input the POUs used in the automation project (that are in PLCopen format) and it generates a XMI file containing the POUs interface. This file is imported into the UML tool (PLCopenToXMI). Three UML elements enriched with a set of stereotypes define a specific POU. In particular, a UML class with *POU stereotype* having a properties: the *name*, *pouType* and *language* that are directly captured from the PLCopen model. An interface element with *FormalParameter* stereotype is added for

each type of variable (e.g. local, input, output...). Finally, variables are defined as properties of the interface enriched with a *stereotype* that fix their type. The *class* with *POU stereotype* uses those interfaces that collect its input formal parameters and realizes the interface that collect output formal parameters (step 2).

After defining the overall automation project, its model is exported into a XMI file (step 3). The next step is the generation of the different system automation projects (*XMIToPLCopen*) and the hardware configurations (*XMIToHwModel*).

The *XMIToPLCopen* transformer filters the software engineering information and transforms it in order to obtain the complete automation project in PLCopen format. Two main parts can be distinguished: firstly, the skeleton of automation project in terms of Configuration(s), Resource(s), Task(s), Variable(s) and PouInstance(s) in PLCopen format have to be generated. Secondly, the code of the required POUs has to be included to obtain the final PLCopen model. This latter requires accessing to the POU repository to recover the functionality of the POUs (see step 4). Table 5-1 presents the six transformation templates used by the *XMIToPLCopen*.

**Table 5-1 Summary of XMIToPLCopen transformation**

UML elements+IEC 61131-3	PLCopen XML element
Component + <<Configuration>>	Configuration
Component + <<Resource>>	Resource
Component + <<Task>>	Task
Interface + <<globalVar>>	globalVars
Interface + <<formalParameter>>	Formal parameters of POUs
Class + <<POU>>	POU

Finally, the *XMIToHwModel* processes the Electrical-Electronic Engineering Domain view and separates the hardware configuration of each controller into different XML

files (one for each controller) that follow the model presented in section 3.3.2 (step 5).

Fig. 5-2 illustrates the general scenario and the different steps of the development phases.

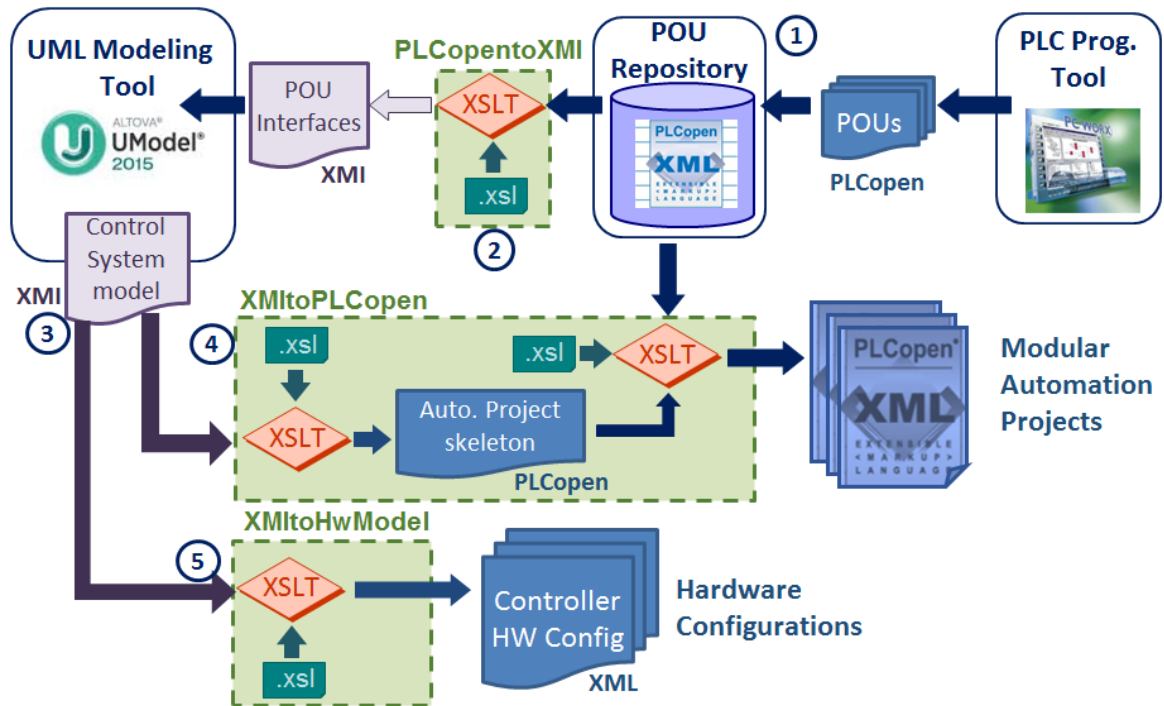


Fig. 5-3 General scenario of the UML modeling tool

## 5.3 Flexible Automation Production System Model

The whole system meta-model from which the generation is derived is illustrated in Fig. 5-4. The system consists of a set of PLCs represented by their automation projects that contain a set of MCs defined (note that a tool to facilitate the modular definition has been presented in Section 5.1). The original MCs are replicated in a number of controllers (*replicated in* in Fig. 5-4). Additionally, each MC is characterized by a set of critical intervals defined by a *condition* the MC variables meet. The condition is expressed as logical expressions of MC variables. The critical intervals prevent reconfiguring the automation system except in case of controller failure. Under a controller failure, it is necessary to diagnose if the automation system is

unrecoverable (a safe stop action is required) or its execution can be resumed at a known previous state (checkpoint), possibly after executing recovery actions (*action* and/or *checkpoint*).

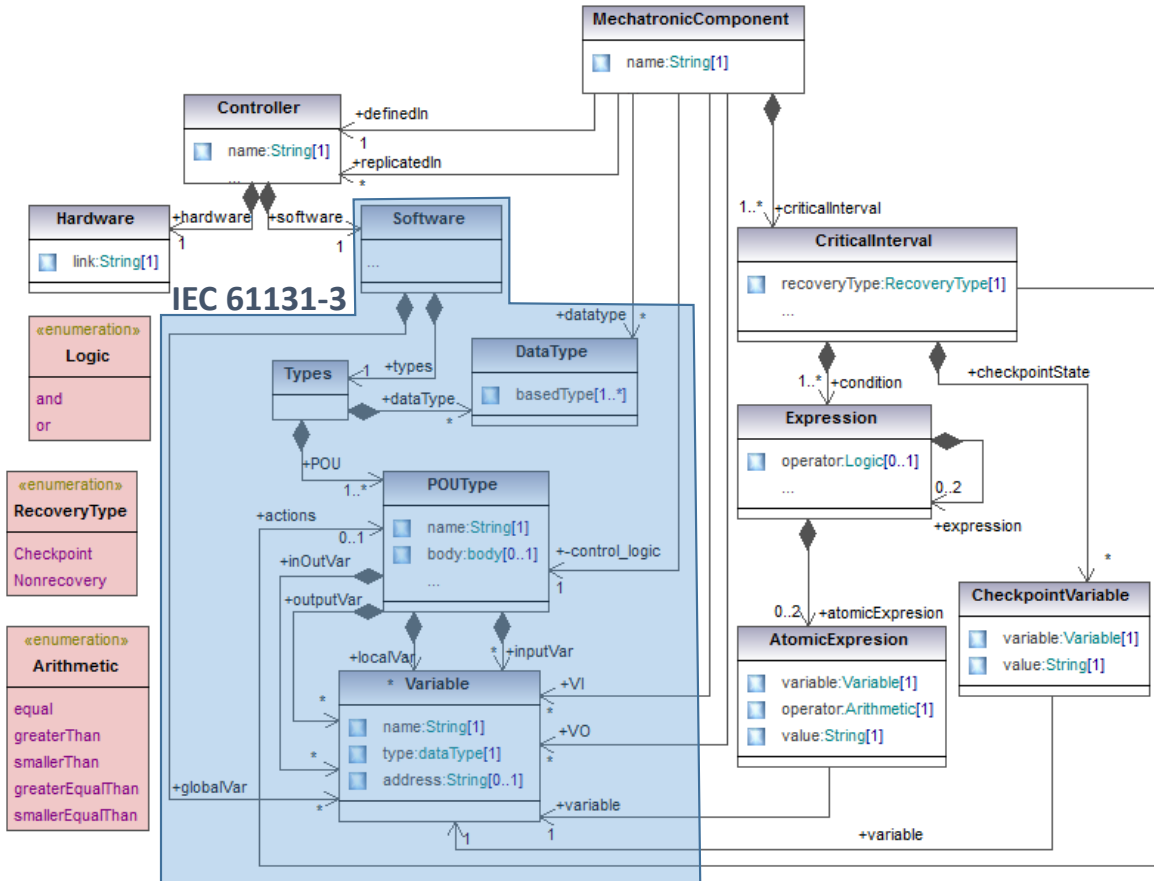


Fig. 5-4 MC Meta-Model

This information is fundamental in order to enable dynamic reconfiguration. Thus, MCAs use this information for diagnosis purposes before starting a reconfiguration. For instance, if a new controller joins the system and transference of workload must be performed, it has to be done at instants in which the production process not be affected, i.e. the system execution state must not belong to a critical interval.

## 5.4 AML-based System definition (FAPS Editor)

The FAPS Editor implements the meta-model of Fig. 5-4 in Automation ML. This standard uses: (1) COLLADA (Anon, 2011) files as a way of defining the geometry and

kinematic information; (2) PLCopen XML files (Marcos et al., 2009) to store the control logic information, and finally (3) the Computer Aided Engineering eXchange (CAEX) (Fedai & Drath, 2005) as the technical basement for the top level format (topology). CAEX is the backbone of the AML standard, since it provides flexible syntactic mechanisms for defining specific semantics and structure of an automation system. These semantics also incorporate information described in other files (COLLADA, PLCopen, among others), which provide a relationship between the different elements of the automation system.

Specifically, the proposal is to follow the guidelines in (Estévez & Marcos, 2012) to implement the meta-model of the system MCs.

- The *System Unit Class Library* defines the concepts involved in the definition of a flexible automation system: It is composed by a series of System Unit Classes (SUC), which represent the elements of the meta-model. Each SUC is characterized by its attributes. If an element is composed of other elements, the complex SUC can contain an Internal Element (IE), which is an instance of another previously defined SUC. Fig. 5-5 depicts the case of the MC definition.
- The *Interface Class Lib* offers a set of interfaces, which allow a SUC or an IE to be linked to an element on an external file. By itself, AML provides a PLCopen interface that provides access to the different elements (POUs and variables) contained in a PLCopen automation project. A new interface, the hardware interface, enables the definition of the controllers and automation projects in PLCopen.
- The *Role Class Lib* contains a series of roles that define the different structure possibilities of the model elements (choice, sequence, all and any roles), which are characterized by two attributes (minOccurs and maxOccurs) that fix their multiplicity.

A more complete explanation of each of these libraries and their corresponding concepts can be found in (Estévez & Marcos, 2012).



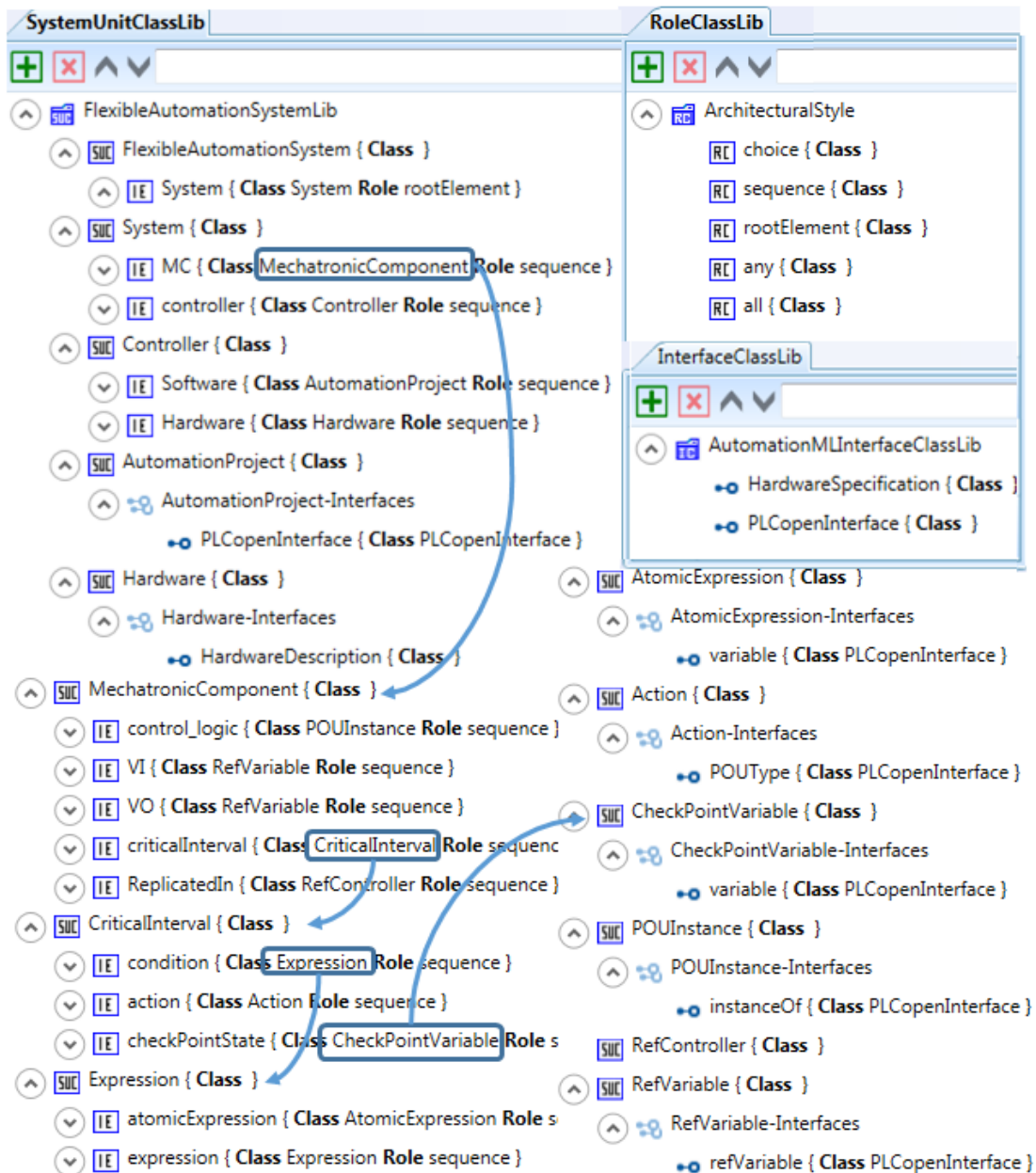


Fig. 5-5 CAEX libraries for Flexible Automation Systems

These libraries are integrated into the AML editor (see *FlexAutSyst SUC library* in Fig. 5-5) allowing the definition of the flexible automation system model. Two external files corresponding to the modular automation project (software) and the hardware description file (hardware) characterize each controller. The basic attributes of the MCs are their name and the controller in which the control logic is described. Additionally, links to POUs and global variables define the control logic of a MC. Finally, the critical intervals are defined making use of expressions involving MC



variables. Fig. 5-6 shows an example of the use of the editor that defines a mechatronic component named MC1.

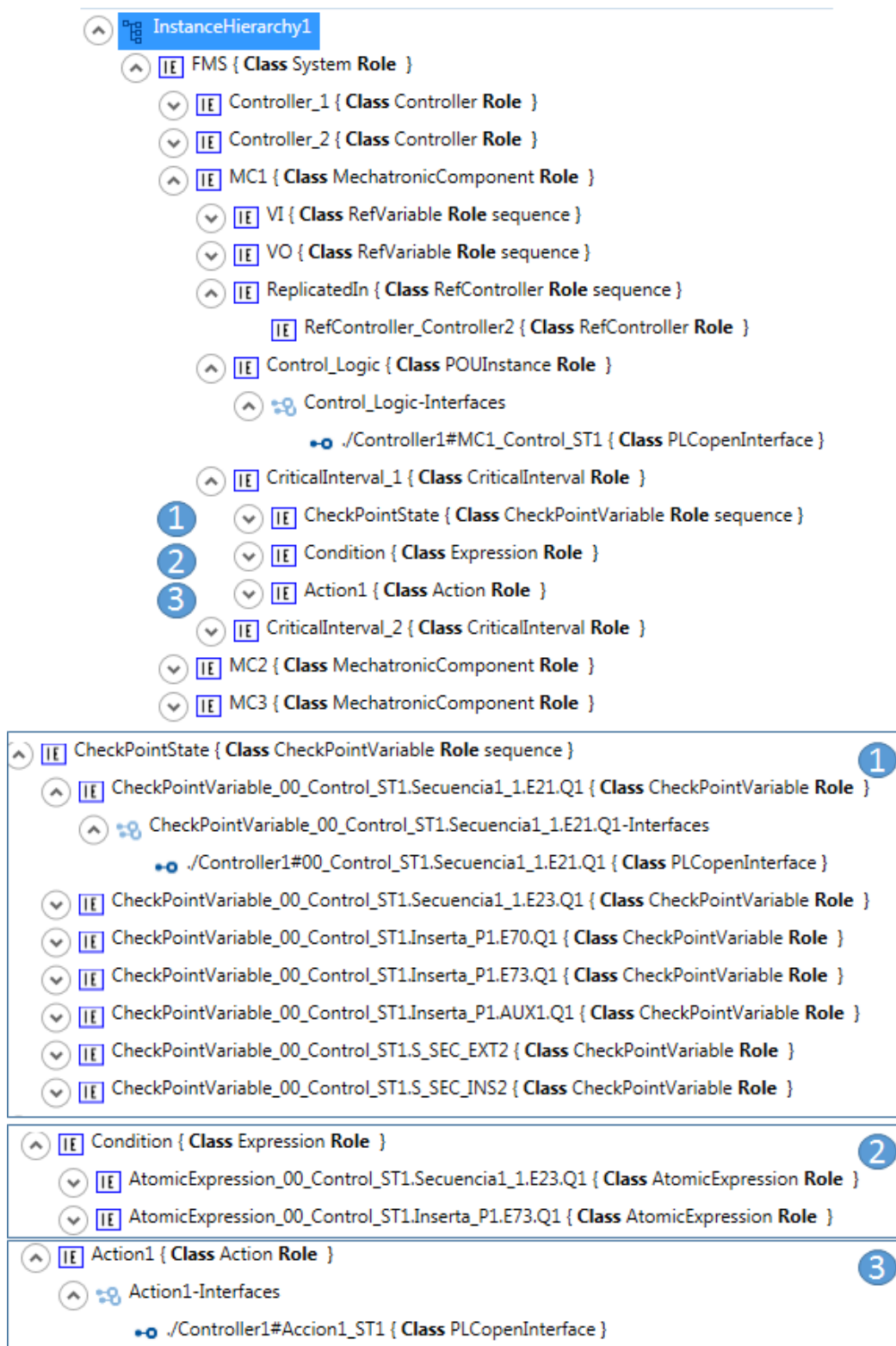


Fig. 5-6 Flexible Automation control system design example

On the other hand, the FAPS editor offers means for characterizing the critical intervals of MCs, defining the expressions relating MC variables that identify checkpoint or unrecoverable critical intervals as well as the recovery actions and the checkpoint state (values of the variables that define the checkpoint state) if needed. The expressions evaluate to a Boolean expression composed by a set of arithmetic and logical operations of some state variables values. Fig. 5-6 uses the AML editor for defining the expression:

“(Control\_ST1.Sequence1\_1.E23.Q1=1 AND Control\_ST1.Insert\_P1.E73.Q1=1)”.

## 5.5 Flexible Automation Projects

Due to current IEC 61131-3 standard execution environments do not allow dynamic code deployment, the generated automation projects contain all MCs the controller can eventually run. This explains why a reconfiguration means de-activation of an MC in a controller and its activation in another.

On the other hand, in order to manage MC execution, MC code is enriched with a wrapper, which allows the MCA to activate/deactivate its execution. Additionally, the current state of the MC (current values of its variables) needs to be read or written by the MCA.

The interaction between the MCA and its corresponding MC is performed via specific sections of the controller memory. This requires the use of specific libraries, provided by the corresponding vendors. For instance, an external access to Beckhoff controllers at run time can be done using ADS (Automation Device Specification) (Beckhoff, 2016). In the same way, Siemens manufacturer provides *libnodave* and *s7netplus* libraries to an external access of S7-300 stations at runtime (Hergenbahn, 2014; Heiser, 2013).

In summary, the Flexible Automation Projects contain not only the functionality of the MCs but also the control code that provides their flexibility. Hence, each MC is composed of three different POUs:

- a) *MCid Control*: a program that manages the execution of the MC.
- b) *MCid Serialize*: a program that collects and serializes the value of the variables that compose the state of the MC;
- c) *MCid Deserialize*: 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.

### 5.5.1 MCid\_Control

This program allows the MCA to activate/deactivate the execution of the logic and recovery/stop actions of the corresponding MC. In fact, this POU acts as a wrapper and provides to MCA an external access required to manage the execution of the MC.

The program interface is a set of application dependent variables and other local static variables that allow MCA to manage it (Table 5-2). These local variables are:

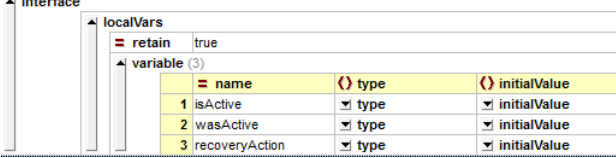
- *isActive* and *wasActive*: boolean variables for managing the activation/deactivation of the MC.

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, Table 5-2 illustrates the general structure and the templates to be applied by the generator.

**Table 5-2 General structure of MCid\_Control program**

Section	Description		
<b>Interface</b>	 <p style="text-align: center;">+ Actions (POU instance variables)</p>		
<b>Body</b>	<b>General structure</b>		
	<pre> IF isActive=TRUE and wasActive=TRUE THEN   MCid();   MCid_Serialize(); ELSE IF isActive=TRUE and wasActive=FALSE THEN   CASE recoveryAction OF /* the list value depends on the MC*/     /* <b>the list cases depends on the critical intervals of the MC</b> */   END_CASE ELSE IF isActive=FALSE and wasActive=TRUE THEN   wasActive=FALSE; END_IF </pre>		
	<b>Direct recovery</b>	<b>Check Point Recovery</b>	<b>Save Stop</b>
<pre> MCid_Deserialize(); MCid (); MCid_Serialize(); wasActive=TRUE; </pre>	<pre> MCid_Action_id(); IF(MCid_Action_id.end) THEN MCid_Deserialize();   MCid ();   MCid_Serialize();   wasActive=TRUE;   recoveryAction=0; END_IF </pre>	<pre> MCid_Action_id(); IF(MCid_Action_id.end) THEN   recoveryAction=0;   isActive=FALSE; END_IF </pre>	

To generate MCid\_Control program from FAPS model three main transformation rules are required:

- Rule 1: Interface definition. This transformation rule is applied to every *InternalElement* having *RefBaseSystemUnitPath* property with *MechatronicComponent* value. First, it adds the common part with the fixed

local variables with their initial values. Then, it adds as many POU instance variables as actions defined in the *CriticalInterval* elements of the MC.

- To add POU Instances, this transformation rule search for those inherited *InternalElements* having *RefBaseSystemUnitPath* property with *Action* and gets the value of its *ExternalInterface*. This will be the type of the new added variable. The name will be the same as the *InternalElement*'s name.
- Rule 2: Body. This transformation rule is applied to every *InternalElement* having *RefBaseSystemUnitPath* property with *MechatronicComponent* value. First, adds the common minimal structure (see Table 5-2). Note that *MCid* is the name of the *ExternalElement* in a *POUInstance*. Furthermore, this template applies Rule 3 to complete the list of the possible cases when one MC is activated.
- Rule 3: Recovery Actions. This transformation rule is applied to every *InternalElement* having *RefBaseSystemUnitPath* property with *CriticalInterval* value. According to the value of *recoveryType* property, the code to add varies (see Table 5-2). Note that, the *MCid* is customized following the procedure commented above.

Fig. 5-7 illustrates the generation process for the control program corresponding to a MC (*MC1\_ST\_Control*). The left side of the figure presents the flexible model that defines the flexible manufacturing system and the right one presents the corresponding control program, *MCid\_Control*, in PLCopen XML format.

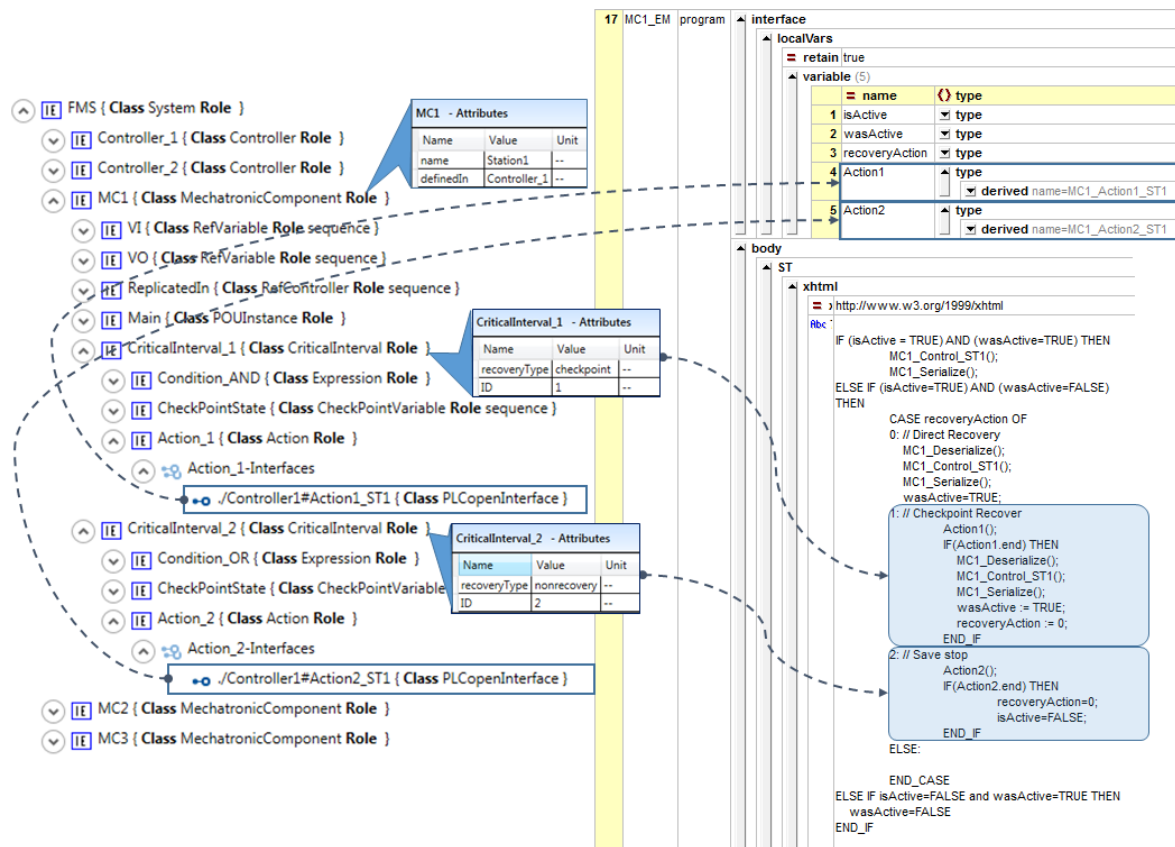


Fig. 5-7 Execution control program generation example.

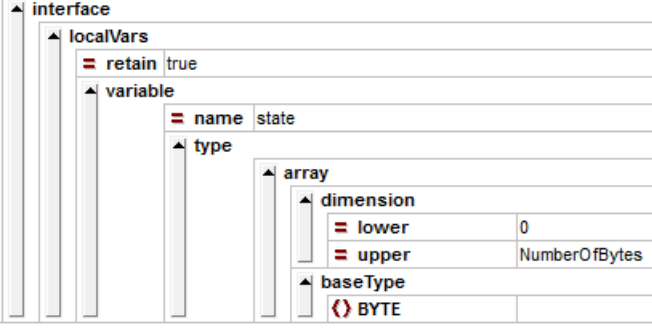
## 5.5.2 Serialization and de-serialization of the MC's state

The serialization program is able to gather the execution states into a byte array, which is accessible by the MCA. The use of byte array is due to most IEC 61131-3 environments offer transformation functions that encompass all other data types (*TypeOfVariable\_TO\_BYTE*), e.g. *BOOL\_TO\_BYTE*, *WORD\_TO\_BYTE* ...

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.

Table 5-3 illustrates the general structure and the templates the generator customizes in order to generate *MCid\_Serialize* and *MCid\_Deserialize* programs.

Table 5-3: General structure of *MCid\_Serialize* and *MCid\_Deserialize* programs

Section	Description
<b>Interface</b>	
<b>Body</b>	<p><b>Serialize</b></p> <pre>state[0] := TypeOfGlobalVariable_TO_BYTE(GlobalVariable); ... state[NumberOfBytes] :=     TypeOfGlobalVariable_TO_BYTE(GlobalVariable);</pre> <p><b>De-serialize</b></p> <pre>GlobalVariable=BYTE_TO_TypeOfGlobalVariable(state[0]); ... GlobalVariable = BYTE_TO_TypeOfGlobalVariable (state[NumberOfBytes]);</pre>

Three main transformation rules are required:

- **Rule1: Interface Definition.** This transformation rule is applied to every *InternalElement* having *RefBaseSystemUnitPath* property with *POUInstance*. First, it computes the number of bytes required for defining the state (*NumberOfBytes* of Table 5-3). To do this, the POU that implements the control logic of MC is analysed to look for local and global variables and input and output parameters. This POU is located in the name of the *ExternalElement*. Finally, this transformation rule applies the next rules (Rule 2, Rule 3) for generating the body of *MCid\_Serialize* or *MCid\_Deserialize*, respectively.
- **Rule 2: Serialize Body.** This transformation rule requires the MC *state* and the variables that compose it (see Table 5-3).

- **Rule 3: De-Serialize Body.** This transformation rule requires the *state* and the variables related to it. The result of the transformation is the writing of the new state (see Table 5-3).

It is important to note, that not all PLC environments allow changing the interval variables of a POU from an external program, preventing the use of a de-serialization program (like in the case of Beckhoff). In these cases, the de-serialization program is implemented in the MCA.

Fig. 5-8 illustrates an example of a Flexible Automation Project that contains the POU, data types, global variables and tasks associated to three MCs (MC1- MC3).

The figure shows a project structure with two main panes: 'project' and 'instances'.

**Project Structure:**

- fileHeader: companyName=Beckhoff Automation GmbH productName=Tw...
- contentHeader: name=controller1
- types
  - dataTypes
  - pous (64)
 

#	name	pouType	interface	body
1	MC1_Control_ST1	program	interface	body
2	MC1_Extract_Base	functionBlock	interface	body
3	MC1_Lack_Base	functionBlock	interface	body
...	...	...	...	...
13	MC1_Action1_ST1	functionBlock	interface	body
14	MC1_Action2_ST1	functionBlock	interface	body
15	MC1_Serialize	program	interface	body
16	MC1_Deserialize	program	interface	body
17	MC1_EM	program	interface	body
18	MC2_Control_ST2	program	interface	body
...	...	...	...	...
31	MC2_Action1_ST2	functionBlock	interface	body
...	...	...	...	...
37	MC2_Serialize	program	interface	body
38	MC2_Deserialize	program	interface	body
39	MC2_EM	program	interface	body
40	MC3_Control_ST3	program	interface	body
...	...	...	...	...
58	MC3_Action1_ST3	functionBlock	interface	body
...	...	...	...	...
62	MC3_Serialize	program	interface	body
63	MC3_Deserialize	program	interface	body
64	MC3_EM	program	interface	body

**Instances:**

- configurations
  - Configuration1
    - resource
      - na... Resource1
        - task (3)
 

#	inter	name	interval	pouInstance
1	PT0.01S	Station1	22	pouInstance name=M...
2	PT0.01S	Station2	20	pouInstance name=M...
3	PT0.01S	Station3	24	pouInstance name=M...
        - globalVars
          - Global\_Variables
            - variable (98)
 

#	name	address	type
1	MC1_START_S1	%MX80.0	type
2	MC1_STOP_S1	%MX80.1	type
...	...	...	...
31	MC2_START_S2	%MX84.0	type
32	MC2_STOP_S2	%MX84.1	type
...	...	...	...
53	MC3_START_S3	%MX90.2	type
54	MC3_STOP_S3	%MX90.3	type
...	...	...	...

**Legend:**

- From the Modular Automation Project (pink, blue, orange)
- Automatically generated (green)

Fig. 5-8 Example of Flexible Automation Project

## 5.6 Application Agents

The Flexible Automation Framework generates CAs and MCAs corresponding to the different controllers and MCs in the system. It is also in charge of generating the MC



diagnosis files, which contain the critical intervals used by the MCA to diagnose the MC state.

### 5.6.1 MC Agent Templates

As established in section 4.5.1 MCAs implement a Finite Machine State consisting of a set of states: *Boot*, *Active*, *Tracking*, *Wait decision* and *End states*. 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. Hence, in order to offer a generic and customizable solution, a MCA Template, illustrated in Fig. 5-9, is proposed. It has two customizable parameters:

- *MC ID*, which is the ID of the corresponding MC.
- *state\_diagnosis*: it addresses the set of masks that define the critical intervals of the MC.

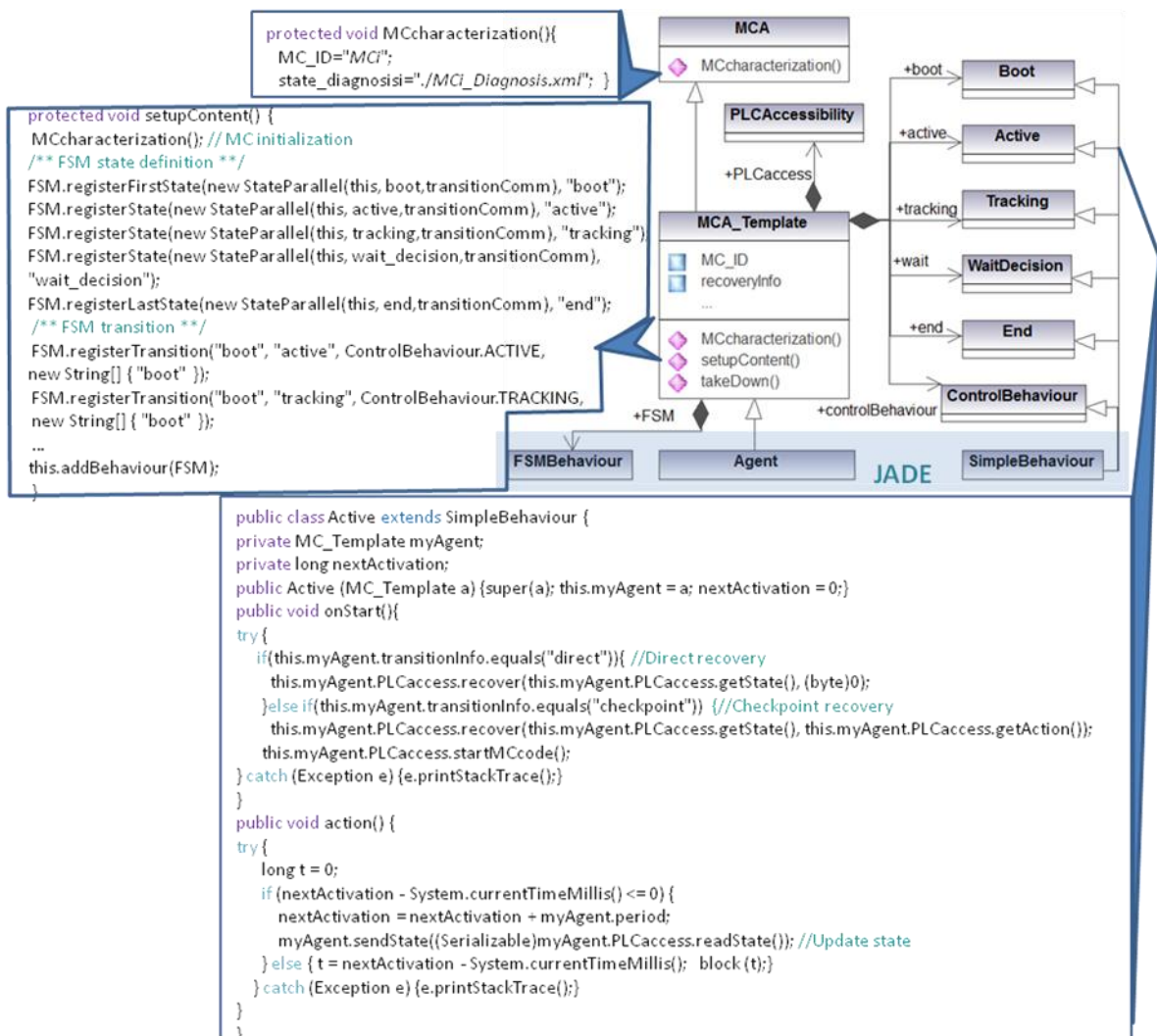


Fig. 5-9 General Structure of System MCAs

Fig. 5-9 depicts the MCA Class Diagram as well as the general structure of the MCA template (*MCA*) that is implemented as a Jade FSM. Each state is associated to the corresponding functionality implemented as two JADE behaviours (*SimpleBehaviour*): one that implements the functionality (*Boot, Active, Tracking, WaitDecision* and *End*) and a second in charge of receiving messages from the middleware agents (*Message\_Queue*). Meanwhile, *MCaccess* provides access to the MC code in the PLC. Fig. 5-9 also shows the skeleton of the *setup* method, responsible for the FMS generation. Additionally, the bottom part of Fig. 5-9 presents the skeleton of the Functionality behaviour for the Active state to illustrate how the default methods of JADE Simple Behaviour, *onStart* and *action* methods are customized for the different states.

## 5.6.2 Controller Agent Template

Following the same philosophy as for MCAs, to offer a generic and customizable solution, a CA Template with a set of customizable parameters is proposed:

- *ID*, which identifies the agent in the system.
- A textual *Description*
- *CPUfactor*, with respect to a reference controller.
- *Memory* resources.
- *AssignedMC*: a list of MCs, whose control logic is executed in the controller.

Fig. 5-10 presents the Class Diagram corresponding to the CA template as well as the parameterization for every CA of the system. The basic functionality of the CA is implemented in a cyclic behavior (*functionality*) in order to process the messages from the middleware agents. These messages correspond to queries about the controller resources or negotiation messages. The reception of a negotiation message provokes the generation of a negotiation behavior that lasts until the negotiation

ends. On the other hand, the CA can implement resource monitoring behaviours that allow monitoring a specific resource of the controller as part of the QoS monitoring process.

The registration of the CA and the creation of resource monitoring and functionality behaviours are performed during the *setup* method of the CA.

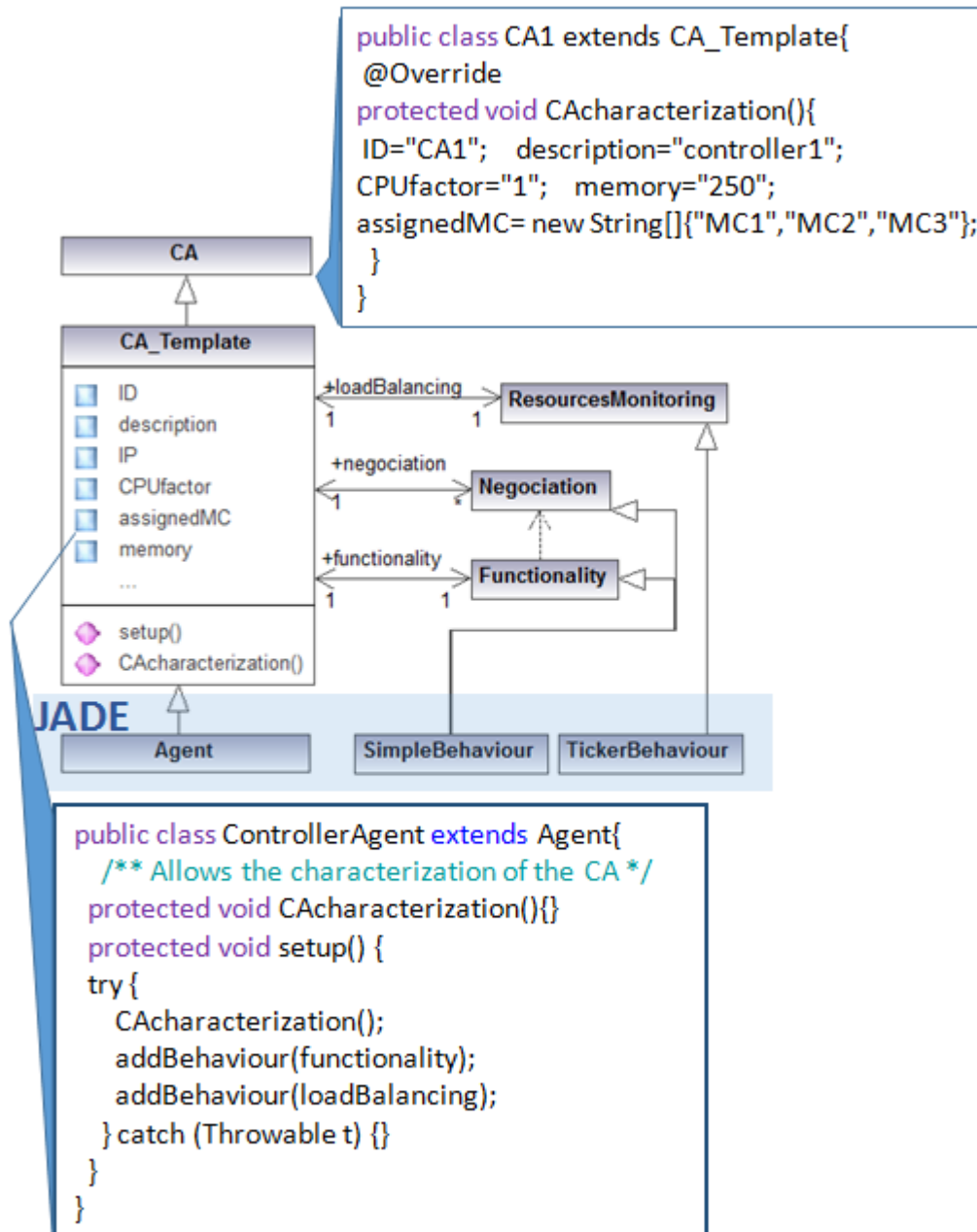


Fig. 5-10 General Structure of System CAs

### 5.6.3 Diagnosis File.

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 having the following structure:

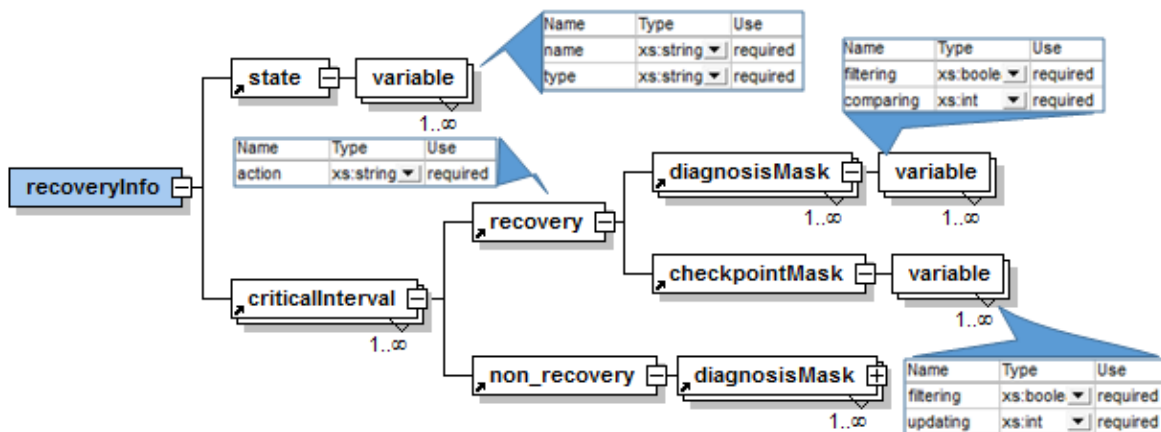


Fig. 5-11 General Structure of Diagnosis.xml files

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.

To generate *MCid\_Diagnosis.xml* file from FAPS model the following transformation rules are required:

- **Rule 1: State characterization** generates the list of variables that form the state. To do this, every *InternalElement* having *RefBaseSystemUnitPath* property with *RefVariable* and the *InternalElement* having *RefBaseSystemUnitPath* property with *POUinstance* in MC are processed.
- **Rule 2: Critical Interval.** This transformation rule applies to every *InternalElement* having *RefBaseSystemUnitPath* property with *CriticalInterval*

in a MC and generates the Critical Interval information stored in diagnosis XML file.

- Rule 3: Diagnosis. This rule processes the Condition to obtain the list of the state variables involved in the condition. This rule applies to every *InternalElement* having *RefBaseSystemUnitPath* property having an *Expression* in a *CriticalInterval*.
- Rule 4 Checkpoint. This rule processes the *CheckpointState* from which the MC can be restarted. This rule applies to every *InternalElement* having *RefBaseSystemUnitPath* property with *CheckPointVariable* in a *Critical Interval*.

## 5.7 Conclusions

The section has presented a series of model-based tools used to help in the definition of a flexible automation system. The first of these tools focuses on the definition of a modular automation system; while the second one is used to add flexibility to the automation system. Automatic code generation is used to generate extended MC control code able to reconfigure MC execution as well as diagnosis information and application agents used at runtime to decide reconfiguration instants.

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.



## **6 PROOF OF CONCEPT**

---



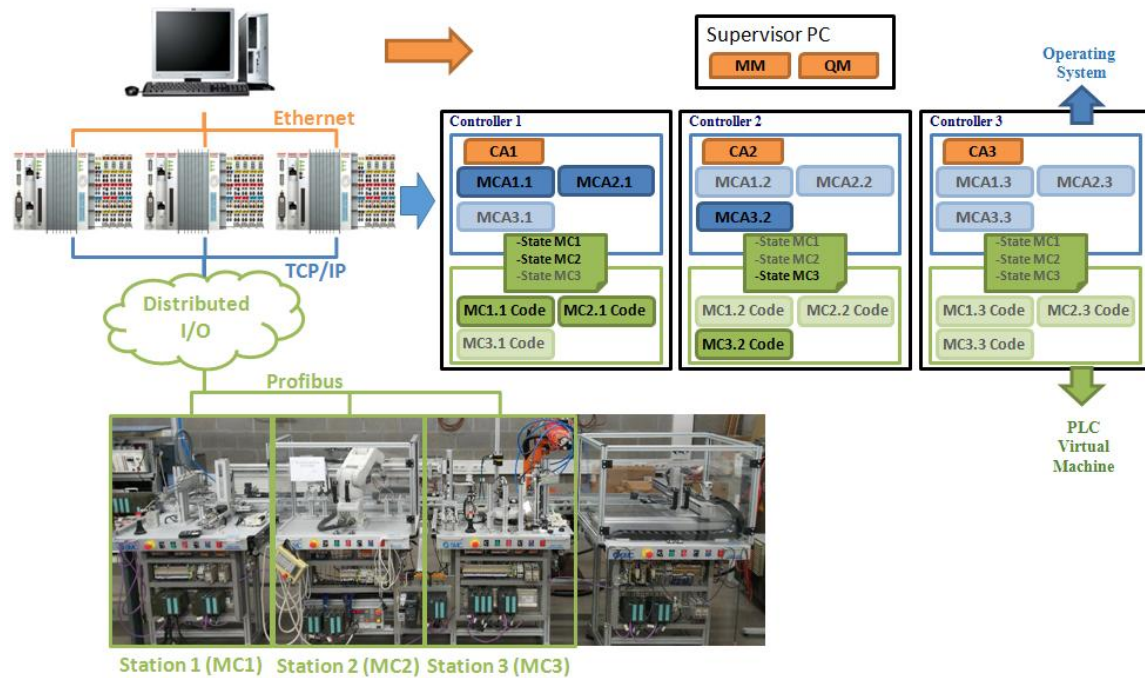


## 6.1 Introduction

This chapter focuses on the validation of the Flexible Automation Framework (FAF) through a case study, as well as the customization of the Flexible Automation Middleware (FAM) for two different QoS (availability and load balancing). The run-time performance of the FAM is also evaluated.

## 6.2 Manufacturing Demonstrator

The prototype of the architecture proposed in this work has been applied to the flexible assembly cell FMS-200, located in the Department of Automatic Control and System Engineering of the Basque Country University. As it is depicted in the bottom part of the Fig. 6-1, the 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 is fed, its orientation is verified and, if correct it is moved to the pallet located on the transfer system. In the second one, the bearing and shaft are placed on the base, and the cap is put in the third station. In the final station, the set mounted on the pallet is stored in the warehouse.



**Fig. 6-1 Manufacturing System Demonstrator**

The cell is divided into five MCs, one for each of the four stations and one 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.

As illustrated in Fig. 6-1 the three MCs have been assigned to three different Beckhoff CX1020 controllers. These are commercial off the shelf Programmable Automation Controllers (PAC), which run a Windows XP embedded operating system in parallel with the Beckhoff PLC runtime. The Controller Agents (CAs) and Mechatronic Component Agents (MCAs) associated to MCs will execute in the operating system, while the control code of MCs will execute in the PLC runtime. The demonstrator is also equipped with a supervisor PC in which the Middleware Manager and the QoS Manager will run (see Fig. 6-1).

**Table 6-1 critical intervals of Station 1**

<b>Station 1</b>		
<b>Condition</b>	<i>Control_ST1.Sequence1_1.E23.Q1=1 AND Control_ST1.Insert_P1.E73.Q1=1</i>	
<b>Recovery Action</b>	<i>Action1_ST1: Remove the base from the station</i>	
<b>Critical Interval 1:</b> Placement of the base in the station is unknown	<i>Control_ST1.S_SEC_EXT2=1</i>	
	<i>Control_ST1.S_SEC_INS2=0</i>	
	<b>Checkpoint:</b> Extract a new base	<i>Control_ST1.Sequence_1.E21.Q1=1 Control_ST1.Sequence_1_1.E23.Q1=0 Control_ST1.Insert_P1.E70.Q1=1 Control_ST1.Insert_P1.E73.Q1=0 Control_ST1.Insert_P1.AUX1.Q1=0</i>
	<b>Condition</b>	<i>(Control.Sequence1_1.E23.Q1=1 AND Control.Insert_P1.E74.Q1=1) OR (Control.Sequence1_1.E23.Q1=1 AND Control.Insert_P1.E75.Q1=1)</i>
	<b>Stop Action</b>	<i>Action2_ST1: return component to initial positions and inform the operator</i>
	<b>Critical Interval 2:</b> Base may be obstructing the	

In first station (MC1) the placement of the base is done by a pneumatic actuator which grips the base and moves it from the station to the pallet. This movement is considered a critical operation since it must be executed without interruptions. If a reconfiguration is performed during this operation, different actions need to be taken

in order to re-start the execution, depending on where the base falls. This leads to two critical intervals: the first one covers the initial lifting of the base from the station. The loss of control at this point requires the base to be removed from the station and a new one to be introduced. The second interval encompasses the rest of the movements from the station to the pallet. The loss of control during this interval causes the base falling into the system and blocking it, so the system must be stopped and the operator should be informed of the failure. Table 6-1 presents the different critical interval actions and/or checkpoint associated to the station.

On the other hand, the third station (MC3) insets the lid on the piece. The different operations performed by this station are distributed around a turning table, which leads to multiple control sequences executing in parallel. There are different lids that can be placed into the assembly. These lids differ on the material, colour and height. The introduction of lids into the plate, the extraction of those that do not correspond to the current assembly and the placement of the lid are considered critical operations. Therefore, the critical interval encompasses all the parallel execution state of the control sequence. The station has five critical intervals, with their corresponding actions that allow the station to continue extracting the lids from the warehouse.

On the other hand and as commented above, current industrial communication protocols do not allow dynamic assignment of inputs and outputs (sensors and actuators) to a controller during execution. This prevents moving MCs between controllers. In order to solve this technological problem, the demonstrator uses a unique PROFIBUS master for all the distributed I/Os. This master is responsible of transferring MC input data to the corresponding controller and the received data from the controllers to MC outputs. The distributed I/O master is a S7-300 Siemens PLC connected to the Beckhoff controllers using an Ethernet connection and having a set of Profibus slaves (MC I/Os).

The distributed I/O reads the input variables from the Profibus bus and sends them to the different controllers using a broadcast message. Each controller takes the information corresponding to their MCs and writes it down into its input variables. In the same way, controllers take the value of the output variables and send them to the

master (TCP/IP message), in charge of distributing them to the corresponding stations.

## 6.3 Modular Automation System

The generation of modular automation system follows the methodology presented in Chapter 5 which allows the definition of modular automation projects as well as the hardware components of the controllers in the system.

The control logic for station 1 and 2 have been defined as part of the automation project related to *Controller1*. Meanwhile, the control logic of station 3 is defined in *Controller2*. For the case of this demonstrator, *Controller3* is defined as an empty automation project, used for illustrating the work balance QoS reconfiguration.

Fig. 6-2 illustrates part of the automation project for *Controller1* in PLCopen format. In particular, it depicts the POU, configuration and resource related to the first and second station.

The screenshot displays the PLCopen project editor interface, divided into two main panes: 'project' on the left and 'instances' on the right.

**Project Pane (Left):**

- types**
  - dataTypes**
    - 1 WAREHOUSE\_S2 (baseType)
  - pous**

#	name	pouType	interface	body
1	Control_ST1	program	interface	body
2	Sequence_ST1	functionBlock	interface	body
3	Ini_ST1	functionBlock	interface	body
4	Extract_Base	functionBlock	interface	body
5	Lack_Base	functionBlock	interface	body
6	Insert_P	functionBlock	interface	body
7	Verify_P	functionBlock	interface	body
8	Initial_Condition	functionBlock	interface	body
9	Control_Command	functionBlock	interface	body
10	SquareWaveGen	functionBlock	interface	body
11	Remove_Pallet	functionBlock	interface	body
12	Place_Pallet	functionBlock	interface	body
13	Action1_ST1	functionBlock	interface	body
14	Action2_ST1	functionBlock	interface	body
15	Control_ST2	program	interface	body
16	Sequence_ST2	functionBlock	interface	body
17	Ini_ST2	functionBlock	interface	body

**Instances Pane (Right):**

- configurations**
  - Configuration1
    - resource
      - task (2)
        - 1 Station1 PT0.01S/22 (pouInstance name=Control\_ST1)
        - 2 Station2 PT0.01S/20 (pouInstance name=Control\_ST2)
    - globalVars
      - Global\_Variables
        - variable (52)

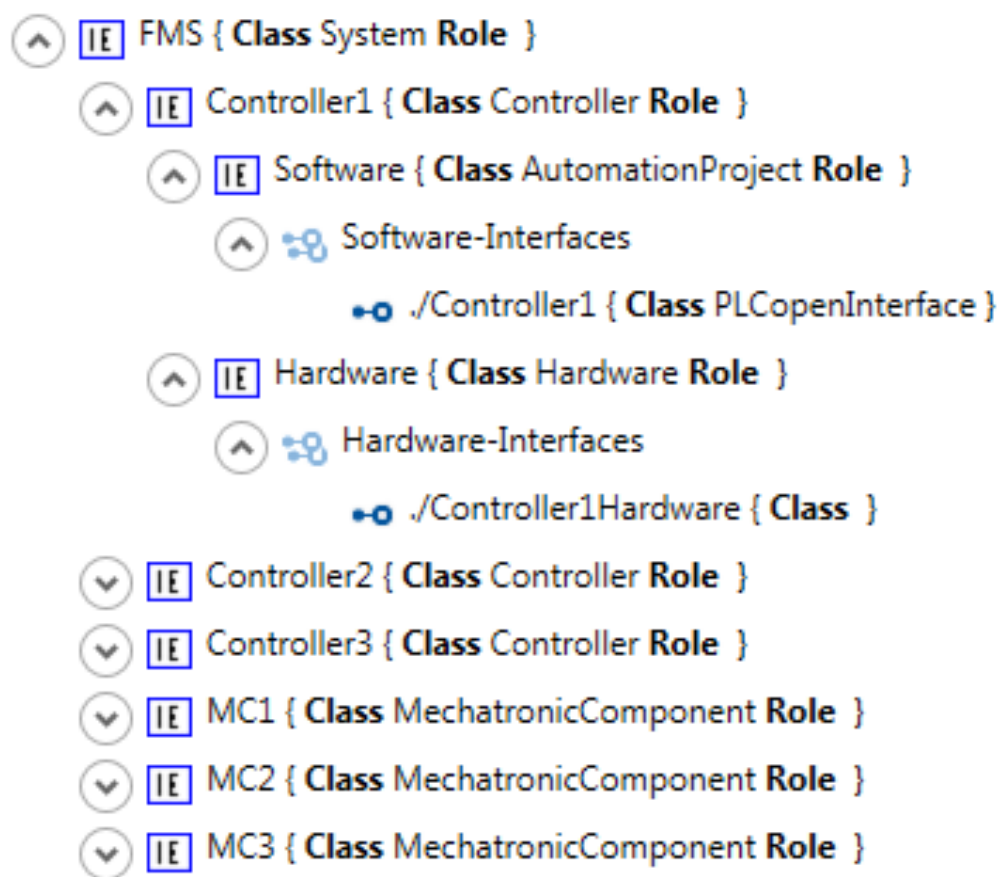
#	name	addr...	type
1	START_S1	%MX80.0	type
2	STOP_S1	%MX80.1	type
...	...	...	...
19	ALARM_S1	%MX180.0	type
20	A_mas_S1	%MX180.1	type
...	...	...	...
31	START_S2	%MX84.0	type
32	STOP_S2	%MX84.1	type
...	...	...	...
44	ALARM_S2	%MX184.0	type
45	SERVO_ON_S2	%MX184.1	type
...	...	...	...

Fig. 6-2 Modular Automation Project Controller1

## 6.4 Flexible Automation System Design

Following the generation of the modular automation projects, the Flexible Automation Framework (FAF) is used for describing the Flexible Automation Production System (FAPS) and for generating the flexible automation projects, as well as the application agents and system information used by the Middleware.

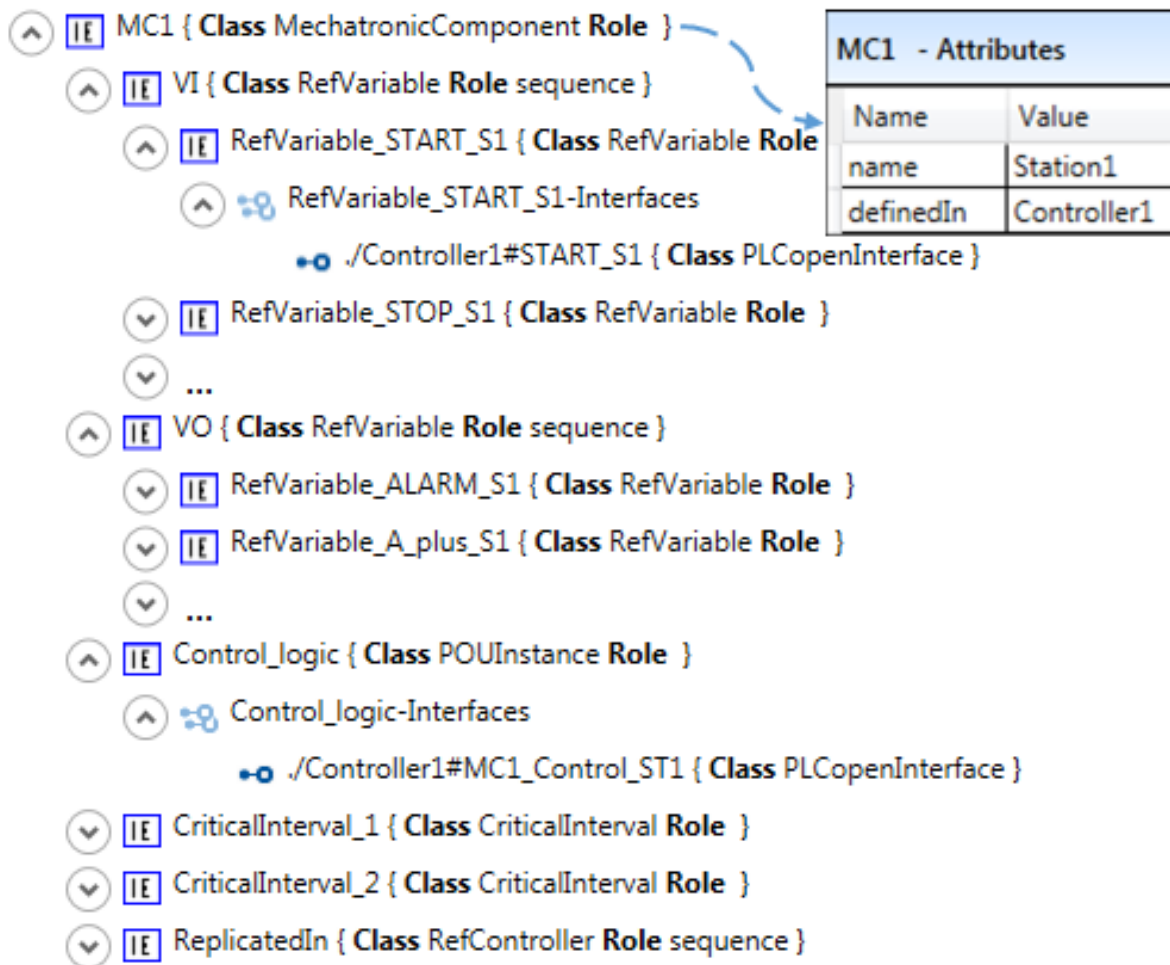
As previously commented in section 5.3 the FAPS model is defined using an AML editor. The controllers are defined by linking them to their corresponding automation project and hardware specification file. Fig. 6-3 illustrates this definition.



*Fig. 6-3 AML controller definition*

After defining the controllers in the system, MC information is defined. Each MC is mapped to one of the modules of the automation project. The control logic, input variables and output variables of a MC are defined as links to the POUs and the

corresponding global variables of the automation project in which they are defined (*defined in*). Fig. 6-4 illustrates this process for MC1.



**Fig. 6-4 definition of MC1 in FAPS Model Editor**

After the MCs, their critical intervals and the controllers in which they may run are specified. For instance, station 1 has two critical intervals, as presented in Table 6-1, which are described using links to variables and POUs contained in the automation project of the controller. Besides, the controllers in which it must be deployed are controller2 and controller3. Fig. 6-5 presents the definition of the critical intervals and replication information of MC1.

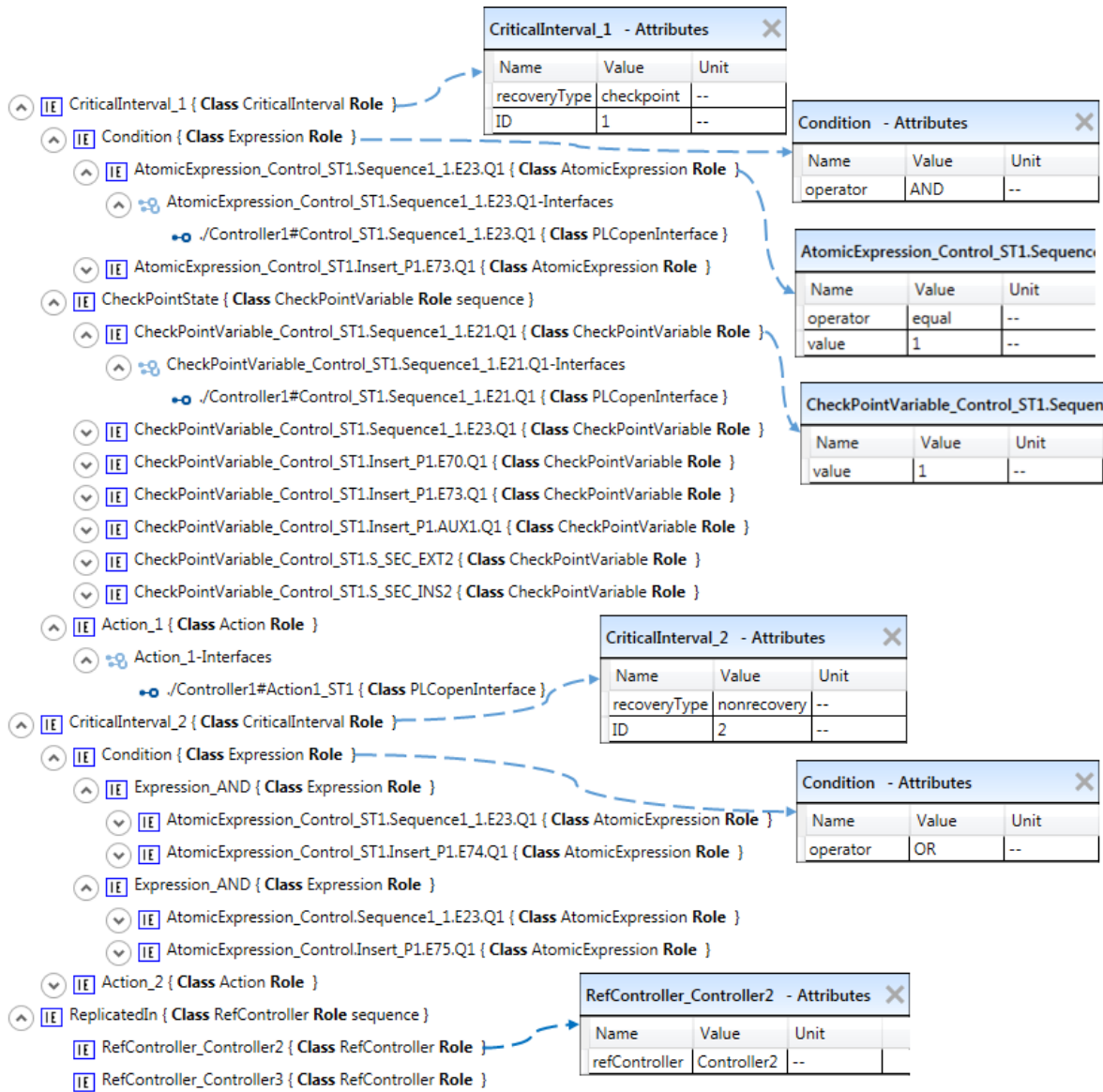


Fig. 6-5 Critical intervals and replication information of MC1

Following the definition of the flexible automation system model, the generation of the automation projects is launched. The results of the transformation are three flexible automation projects that contain the POU, data types and global variables associated to each of the MCs they may run. At the same time, the execution management and serialization programs for each MC are generated. Fig. 6-6 shows part of the flexible automation project to be download on controller1.



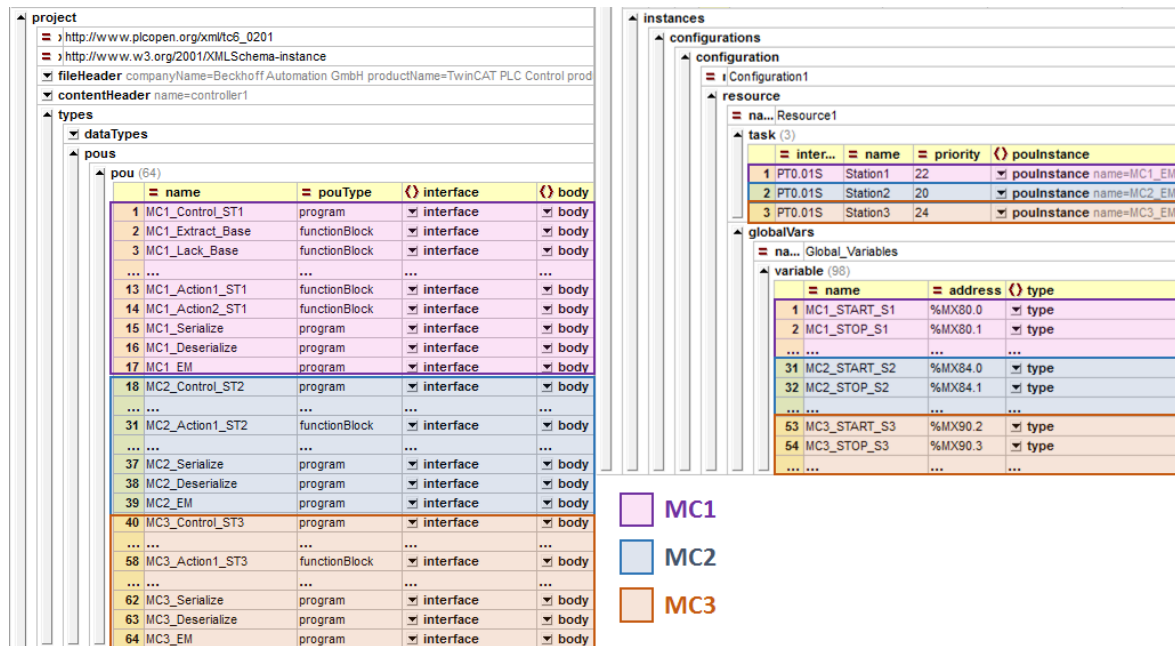


Fig. 6-6 Flexible automation project for controller 1

With respect to the middleware application agents, the FAF generates a Controller agent for each of the three controllers (see example in Fig. 6-7), a MCA for the MCs (see example in Fig. 6-8) as well as their corresponding diagnosis file (see example in Fig. 6-9).

```
public class CA1 extends CA_Template{
    @Override
    protected void CAcharacterization(){
        ID="CA1"; description="controller1";
        CPUfactor="1"; memory="250";
        assignedMC= new String[]{"MC1","MC2","MC3"};
    }
}
```

Fig. 6-7 Controller agent for Controller1

```
public class MCA1 extends MCA_Template{
    @Override
    protected void MCcharacterization(){
        MC_ID="MC!";
        state_diagnosisi="./MCi_Diagnosis.xml!";
    }
}
```

Fig. 6-8 MC agent for MC1



The image shows two side-by-side XML tree views. The left view is titled 'XML' and shows a 'recoveryInfo' section with a 'state' subsection containing a 'variable (305)' table and a 'criticalInterval' subsection with a 'recovery' subsection containing an 'action 1' and a 'diagnosisMask' subsection with a 'variable (306)' table. The right view is titled 'checkpointMask' and shows a 'variable (306)' table and a 'non\_recovery' subsection with an 'action 2' and a 'diagnosisMask (2)' subsection containing two 'variable (306)' entries.

recoveryInfo		
state		
variable (305)		
name	type	
1	.MC1_START_S1	BOOL
2	.MC1_STOP_S1	BOOL
...	...	...

criticalInterval		
recovery		
action 1		
diagnosisMask		
variable (306)		
filtering	comparing	
1	0	0
...	...	...
98	255	1
...	...	...
206	255	1
...	...	...

checkpointMask		
variable (306)		
filtering	updating	
1	255	0
...	...	...
96	0	1
...	...	...
206	0	0
...	...	...

non_recovery	
action 2	
diagnosisMask (2)	
variable	
1	variable (306)
2	variable (306)

Fig. 6-9 Diagnosis file for MC1

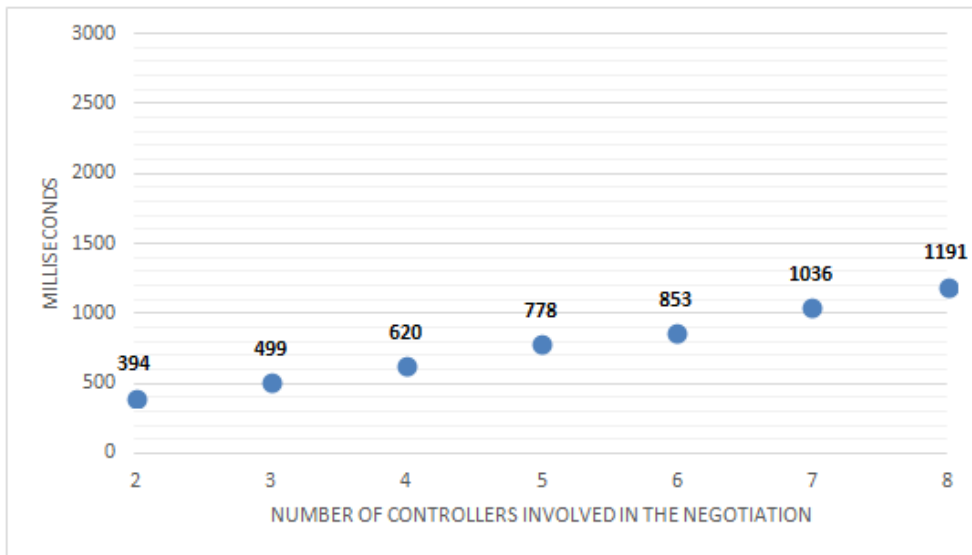
## 6.5 Run-time Performance

The performance of the proposed middleware architecture has been assessed measuring the time it takes the FAM to recover the QoS in the case of availability and system efficiency.

### 6.5.1 Availability QoS

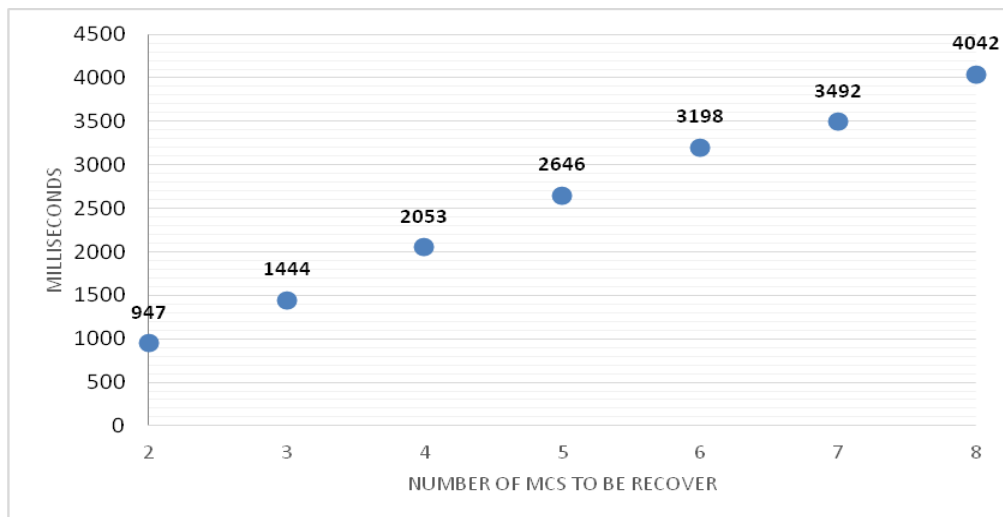
In the case of availability, the recovery time is measured starting from the first QoS loss received by the QM up to the first execution of the MCs in their new locations.

The first test makes a comparison of the recovery time based on the number of controllers involved in the negotiation. The result is shown in Fig. 6-10. As expected, as the number of controllers involved in the negotiation increases the time it takes the QM and the D&D to reach a decision increases. This has led to the conclusion that reducing the number of tracking MCs would be better for a faster recovery although it would reduce the possibilities of redistribution.



**Fig. 6-10 Recovery Time vs. Number of Controllers Involved in the Negotiation**

The second test compares the recovery time based on the number of MCs to be recovered. The result of this test is presented in Fig. 6-11. As anticipated, the time increases based on the number of the MCs, illustrating the scalability of the recovery process.



**Fig. 6-11 Recovery Time vs. Number of MCs to be recovered**

Table 6-2 presents the time it takes for each of the MCs to recover its execution, related to the number of MCs to be recovered. As it can be seen, that recovery time for the MCs is nearly the same, except for the recovery time of the first MC (MC1). This is

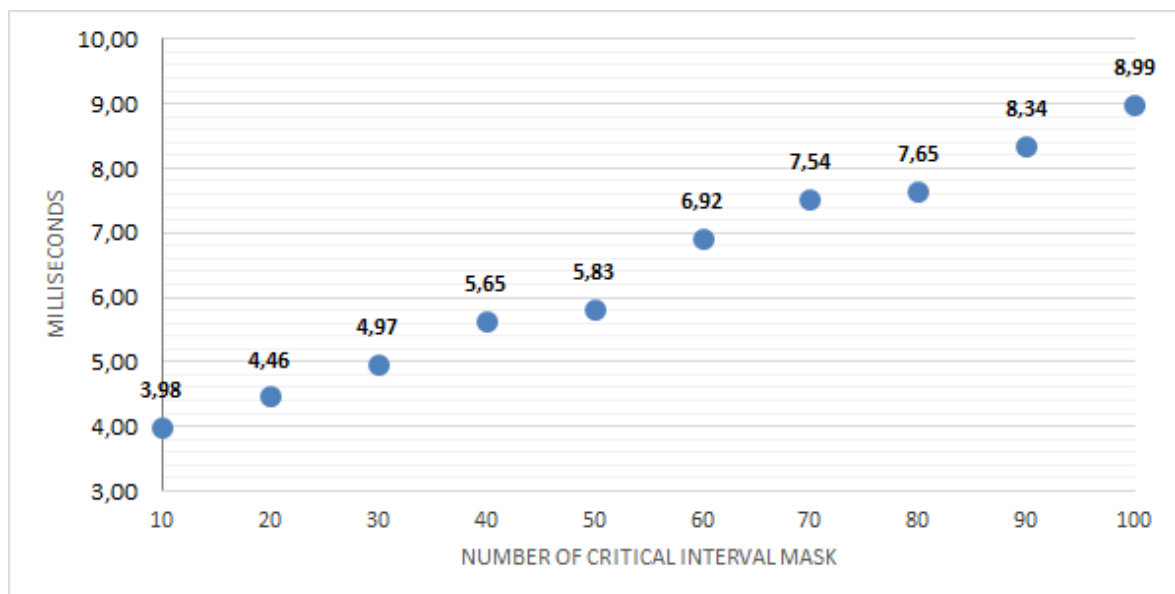
due to this recovery time also contemplates the time it takes the D&D to change all passive MCAs to the *wait decision* state.

**Table 6-2 recovery time for each MCs**

# MCs	MC 1	MC 2	MC 3	MC 4	MC 5	MC 6	MC 7	MC 8
1	401							
2	630	317						
3	828	321	295					
4	998	379	335	341				
5	1194	448	364	320	320			
6	1393	284	425	366	289	381		
7	1603	288	332	330	321	310	308	
8	1761	309	284	419	317	291	354	307

The previous test presents the influence the number of MCs and controllers have over the recovery. However, these tests have been performed in a system with a specific number of critical intervals. To determine the influence the process of finding if the current state belongs to a critical interval has over the recovery time, a new test is performed. This test measures the time it takes the MCA to diagnose the execution state based on the complexity and number of the critical interval expressions (number of masks).

The results of the test are presented in Fig. 6-12. As expected, the diagnosis time increases with the number of masks. However, this increase is minimal, making it possible to implement big number of masks in order to diagnose the state.



**Fig. 6-12 Diagnosis time vs Number of critical interval masks**

## 6.5.2 System Efficiency QoS

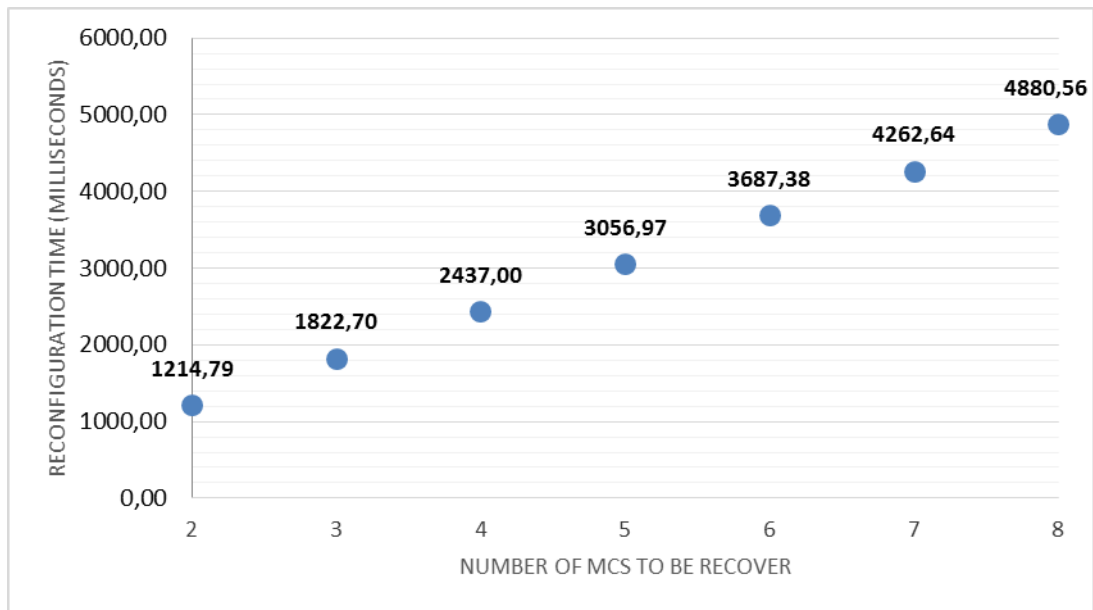
The performance of the middleware in the case of system efficiency is measured using the time it takes the D&D to redistribute the MCs based on the number of MCs to be distributed. For this test a basic redistribution algorithm has been used that could be optimized.

Table 6-3 presents the results of this test. The first column represents the time the D&D takes to calculate the new system distribution; meanwhile, the following columns present the time it takes each of the MCs to relocate.

**Table 6-3 Re-distribution algorithm and reconfiguration times**

# MCs	Re-distribution algorithm	MC 1	MC 2	MC 3	MC 4	MC 5	MC 6	MC 7	MC 8
2	5,29	604,75	604,75						
3	6,14	604,52	605,43	606,61					
4	8,36	603,83	609,32	608,84	606,65				
5	11,67	604,90	620,01	607,06	606,79	606,54			
6	12,72	604,70	605,12	605,07	604,78	633,43	621,56		
7	16,98	608,54	604,80	607,30	606,76	606,37	606,47	605,43	
8	17,49	605,50	619,59	608,67	605,25	606,49	604,55	608,21	604,81

Fig. 6-13 presents a graphical representation of the time it takes the D&D to reconfigure the whole system with respect to the number of MCs.



*Fig. 6-13 Reconfiguration Time vs Number of MCs*

## 6.6 Conclusions

This section has presented the validation of the Flexible Automation Framework (FAF) and the Flexible Automation Middleware (FAM).

To assess the use of the FAF the definition of three stations of a flexible automation cell as MCs has been assumed. This process starts with the different modular automation projects containing the code of the three stations, and shows the definition of the different MCs, critical intervals, and backup controllers, presenting the FAPS Model of the system and the output flexible automation projects and application agents.

From the run-time assessment performed on the FAM, it can be concluded that the middleware introduces an overhead that depends very much on design decisions: number of replicas for MC and number of controllers containing replicas in the system. These do not vary with the size of the MC (control logic) to be managed. Thus, design decisions should be made based on the critical parts of the system, the reliability and flexibility of the control logic and the level of quality assurance needed in the different parts of the manufacturing process.



## **7 CONCLUSIONS AND FUTURE WORKS**

---





## 7.1 Conclusions

The main goal of this work was to assure the fulfilment of a set of QoS within automation production systems. This work has focused on the identification and definition of mechanisms that ensure the fulfilment of the QoS, the implementation of these mechanisms in an agent based system and the generation of tool support by means of the application of model based techniques. The set of tools generated allows the definition and implementation of the management architecture.

The reconfiguration mechanisms presented in the work are based on the relocation of parts of the automation system. Concretely, the automation system is divided in the so called MCs in charge of controlling part of the process. However, as presented in this work, the instant and actions needed to relocate parts of the automation system are influenced by the operational state of the process. To this respect, two types of states have been identified: a state is said non-critical if the MC can be relocated and started from the last known state. Nevertheless, this work also focuses on the recovery of the system in case of a failure during a critical state. This has led to two types of actions: a recovery from previously known state or a safe stop.

This work not only focuses on the information needed for launching the reconfiguration and recovery of the MCs, it also focuses on the mechanisms needed to supervise the fulfilment of the QoS. This has led to the definition of two modules which monitor the fulfilment of the QoS (QoS monitors) and which determine the best way to recover these QoS when they are lost (Diagnosis and Decision).

The proposed architecture and associated mechanisms have been implemented using an agent-based middleware. As commented in the related work chapter, agent based middleware has been commonly used to introduce reconfiguration mechanisms into distributed automation systems. The proposed middleware consists of a series of agents (supervisor and application agents) that allow monitoring QoS, detecting QoS loss and recovery of the QoS. The middleware is presented as a generic architecture

that provides guidelines to include different monitoring and diagnosis functions by extending the agents functionality.

This middleware has been used to ensure the availability of the control system and to optimise the controller efficiency. These cases represent the different reconfigurations and the flexibility the middleware can provide. The real-time performance of these mechanisms shows that it is necessary to find a compromise between the quality of the flexibility of the automation system and the overhead it adds.

This work also contributes on the application of model-based techniques to generate flexible code and customized middleware agents. This contribution includes tool support, an aspect that is fundamental to achieve the adoption of model based techniques in industry.

The main results of the work have been published in 7 international conferences and 3 papers in international Journals, as well as several national conferences.

In particular, related to the definition of the architecture and its implementation, the results are the following:

Preliminary work on the QoS management architecture, which provides controller fault tolerance by relocating the whole control system, as well as the different types of recovery methods, was presented in Priego, R., Armentia, A., Orive, D. & Marcos, M., 2013. Supervision-based reconfiguration of industrial control systems. In *18th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA)*. Cagliari, Italy, pp. 1–4.)

The first drafts of the agent based middleware for assuring QoS was presented in Priego, R., Gangoiti, U., Orive, D. & Marcos, M., 2014. Agent-Based Reconfiguration at Controller Level. In *19th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA)*. Barcelona, Spain, pp. 1–4; Priego, R., Orive, D. & Marcos,

M., 2014. Maintaining the Availability of the Control System in Industrial Automation. In *Agenten im Umfeld von Industrie 4.0*. pp. 15–21).

(Priego, R., Agirre, A., Estévez, E., Orive, D. & Marcos, M., 2015. Middleware-based Support for Reconfigurable Mechatronic Systems. In *2nd Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT)*. Maribor, Slovenia: Elsevier Ltd., pp. 81–86.) This paper introduces the concept of Mechatronic Component (MC) as the code in charge of controlling a part of the process and it analyzes when it is possible to recover its functionality after a controller fault.

The final structure of the management architecture and the middleware are presented in Priego, R., Iriando, N., Gangoiti, U. & Marcos, M., 2017. Agent Based Middleware Architecture for Reconfigurable Manufacturing Systems. *The International Journal of Advanced Manufacturing Technology*, pp.1–20.

Other publications deal with the uses of model based techniques for the definition of the flexible control systems:

In particular, the use of model based techniques for defining and implementing a modular automation production systems was presented in Priego, R., Armentia, A., Estévez, E. & Marcos, M., 2016. Modeling techniques as applied to generating tool-independent automation projects. *At-Automatisierungstechnik*, 64(4), pp.325–340.

The first drafts of the model based definition of the flexible control systems and the development process was presented in Priego, R., Armentia, A., Orive, D., Estévez, E. & Marcos, M., 2014. A Model-Based Approach for Achieving Available Automation Systems. In B. Edward, ed. *19th World Congress of the International Federation of Automatic Control (IFAC)*. Cape Town, South Africa, pp. 3438–3443.

Meanwhile, Priego, R., Armentia, A., Estevez, E. & Marcos, M., 2015. On applying MDE for generating reconfigurable automation systems. In *13th IEEE International Conference on Industrial Informatics (INDIN)*. Cambridge, UK, pp. 1233–1238 presents

the basics of the model-based tool used for the description and code generation of the flexible automation system.

The final model-based framework for the definition and automatic generation of the flexible system is presented in (Priego, R., Estévez, E., Orive, D., Vogel-Heuser, B. & Marcos, M., 2017. A MDE approach for supporting flexible automation. that is under review in *Mechatronics Journal*.

Finally, during the development of this work a publication related to the use of the proposed multi-agent system for providing run-time updates of the automation system code was presented (Priego, R., Schütz, D., Vogel-heuser, B. & Marcos, M., 2015. Reconfiguration Architecture for Runtime Updates of an Automation System. In *IEEE 20th International Conference on Emerging Technologies & Factory Automation (ETF A)*. Luxembourg, Luxembourg,, pp. 1–8). This was the main result of the research stay at AIS- TUM.

As a final remark, this work has fulfilled its main goal of ensuring the fulfilment of the QoS of an automation system by means of a generic and customisable architecture that can be extended to other QoS.

## 7.2 Future works

The development of this work has allowed detecting a series of interesting research lines which can be investigated in the future:

- To define a methodology for the definition and implementation of new QoS assurance. The current work has proposed the management architecture, supervisor agents, and templates for the monitoring and recovery of a QoS, as well as a model based framework for the definition of the flexible automation. The proof of concept prototype has shown which agents are in charge of the monitoring of the QoS implemented and how the functionality of the monitoring and D&D agents has been extended. However, it would be

interesting to define the generic methodology to follow in order to introduce new QoS into the architecture.

- Other interesting line to explore is related to Machine To Machine communication for achieving horizontal integration. The middleware architecture presented here could be adapted to allow the set of plant machines communicating among them in order to decide how to perform manufacturing orders or how to recover from machine failures.



## **REFERENCES**

---





- Andersson, K., Lennartson, B., Falkman, P. & Fabian, M.Å., 2011. Generation of restart states for manufacturing cell controllers. *Control Engineering Practice*, 19(9), pp.1014–1022.
- Anon, 2016. AutomationML. , p.<http://www.automationml.org/>.
- Anon, 2011. COLLADA. , p.<https://collada.org/> last.
- Anon, Functional Application Design for Distributed Automation Systems (FAVA). , p.<https://www.ais.mw.tum.de/en/research/current-rese>.
- Association, E.F. of the F.R., 2012. *Factories of the Future PPP FoF 2020 Roadmap: Consultation document*,
- Babiceanu, R.F.F.R.F. & Chen, F.F.F., 2006. Development and applications of holonic manufacturing systems: a survey. *Journal of Intelligent Manufacturing*, 17(1), pp.111–131.
- Barata, J. & Camarinha-Matos, L.M., 2003. Coalitions of manufacturing components for shop floor agility - the CoBASA architecture. *International Journal of Networking and Virtual Organisations*, 2(1), pp.50–77.
- Barbosa, J., Leitão, P., Adam, E. & Trentesaux, D., 2015. Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution. *Computers in Industry*, 66, pp.99–111.
- Basile, F., Chiacchio, P. & Gerbasio, D., 2013. On the Implementation of Industrial Automation Systems Based on PLC. *Automation Science and Engineering, IEEE Transactions on*, 10(4), pp.990–1003.
- Beckhoff, 2016. Automation Device Specification (ADS). , p.<https://infosys.beckhoff.com/english.php?content=>.
- Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G., 2008. JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology*, 50(1–2), pp.10–21.
- Bellifemine, F., Poggi, A. & Rimassa, G., 2001. Developing multi-agent systems with a FIPA-compliant agent framework. *Software - Practice and Experience*, 31(2), pp.103–128.
- Bergagård, P., 2015. *On restart of automated manufacturing systems*. CHALMERS UNIVERSITY OF TECHNOLOGY.
- Binotto, A.P.D., Wehrmeister, M.A., Kuijper, A. & Pereira, C.E., 2013. Sm@rtConfig: A context-aware runtime and tuning system using an aspect-oriented approach for data intensive engineering applications. *Control Engineering Practice*, 21(2), pp.204–217.
- Blanchet, M., Rinn, T., Von Thaden, G. & de Thieulloy, G., 2014. *Industry 4.0 The new*

- industrial revolution How Europe will succeed* A. Dujin, C. Geissler, & D. Horstkötter, eds.,
- Booch, G., Rumbaugh, J. & Jacobson, I., 2015. *The Unified Modeling Language User Guide (2nd Edition)*, Addison-Wesley Professional.
- Botygin, I.A. & Tartakovsky, V.A., 2014. The development and simulation research of load balancing algorithm in network infrastructures. In *2014 International Conference on Mechanical Engineering, Automation and Control Systems (MEACS)*. IEEE, pp. 1–5.
- Bousbia, S. & Trentesaux, D., 2002. Self-organization in distributed manufacturing control: state-of-the-art and future trends. *IEEE International Conference on Systems, Man and Cybernetics*, vol.5, p.6.
- Brennan, R.W., Vrba, P., Tichy, P., Zoitl, A., Sünder, C., Strasser, T. & Marik, V., 2008. Developments in dynamic and intelligent reconfiguration of industrial automation. *Computers in Industry*, 59(6), pp.533–547.
- Brussel, H. Van, Wyns, J., Valckenaers, P., Bongaerts, L. & Peeters, P., 1998. Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37(3), pp.255–274.
- Cândido, G., Colombo, A.W., Barata, J. & Jammes, F., 2011. Service-oriented infrastructure to support the deployment of evolvable production systems. *IEEE Transactions on Industrial Informatics*, 7(4), pp.759–767.
- Colombo, A.-W., Karnouskos, S. & Mendes, J.-M., 2010. Factory of the Future: A Service-oriented System of Modular, Dynamic Reconfigurable and Collaborative Systems. In *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*. pp. 459–481.
- Colombo, A.W., Karnouskos, S., Mendes, M., Leit, P., Controls, R., Control, S., Acquisition, D., Mendes, J.M. & Leitão, P., 2015. Industrial Agents in the Era of Service-Oriented Architectures and Cloud-Based Industrial Infrastructures. In *Industrial Agents: Emerging Applications of Software Agents in Industry*. Elsevier, pp. 67–87.
- Commission, I.E., 2004. International Standard IEC 61499 Part 1.
- Duffie, N.A. & Piper, R.S., 1986. Nonhierarchical control of manufacturing systems. *Journal of Manufacturing Systems*, 5(2), p.141.
- Estévez, E. & Marcos, M., 2012. Model-Based Validation of Industrial Control Systems. *IEEE Transactions on Industrial Informatics*, 8(2), pp.302–310.
- Estevez, E., Marcos, M., Gangoiti, U. & 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.

- European Commission: Research and Innovation, 2013. *Factories of the Future PPP: towards competitive EU manufacturing*,
- Fay, A., Vogel-Heuser, B., Frank, T., Eckert, K., Hadlich, T. & Diedrich, C., 2015. Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. *Journal of Systems and Software*, 101, pp.221–235.
- FDCML, F.D.C.M., 2002. *FDCML 2.0 Specification*,
- Fedai, M. & Drath, R., 2005. CAEX - A neutral data exchange format for engineering data. *ATP International Automation Technology*, 01/2005(3), pp.43–51.
- Ferber, J., 1999. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence* 1st ed. A. Wesley, ed.,
- Hästbacka, D., Vepsäläinen, T. & Kuikka, S., 2011. Model-driven development of industrial process control applications. *Journal of Systems and Software*, 84(7), pp.1100–1113.
- Heiser, D., 2013. S7netplus. , p.<http://nugetstatus.com/packages/S7netplus>.
- Hergenbahn, T., 2014. LIBNODAVE - Exchange data with Siemens PLCs. , p.<http://libnodave.sourceforge.net/>.
- Hinchey, M.G. & Sterritt, R., 2006. Self-managing software. *IEEE Computer*, 39(2), pp.107–109.
- Huebscher, M.C. & McCann, J. a, 2008. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys*, 40(3), pp.1–28.
- Intelligent Physical Agents, F. for, 2015. Standard FIPA Specifications. Available at: <http://www.fipa.org/repository/standardspecs.html>.
- International Electrotechnical Commission, 2003. IEC International Standard IEC 1131-3 Programmable Controllers, Part 3: Programming Languages.
- Jamro, M., 2014. Automatic generation of implementation in SysML-based model-driven development for IEC 61131-3 control software. In *2014 19th International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, pp. 468–473.
- Kephrt, J.O. & Chess, D.M., 2003. The Vision of Autonomic Computing. *IEEE Computer*, 36(1), pp.41–50.
- Khalgui, M. & Mosbahi, O., 2010. Intelligent distributed control systems. *Information and Software Technology*, 52(12), pp.1259–1271.
- Koestler, A., 1969. *The Ghost in the Machine*, Arkana Books.
- Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G. & Becker, C., 2014. A survey on

- engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17(Part B), pp.184–206.
- Lastra, J., 2004. *Reference mechatronic architecture for actor based assembly systems*. Tampere University of Technology.
- Legat, C., Schütz, D. & Vogel-Heuser, B., 2013. Automatic generation of field control strategies for supporting (re-)engineering of manufacturing systems. *Journal of Intelligent Manufacturing*, pp.1–11.
- Legat, C. & Vogel-Heuser, B., 2014. A Multi-agent Architecture for Compensating Unforeseen Failures on Field Control Level T. Borangiu, D. Trentesaux, & A. Thomas, eds. *Service Orientation in Holonic and Multi-Agent Manufacturing and Robotics*, 544, pp.195–208.
- Leitão, P., 2009. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7), pp.979–991.
- Leitão, P., 2008. Self-Organization in Manufacturing Systems: Challenges and Opportunities. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. pp. 174–179.
- Leitão, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T. & Colombo, A.W., 2016. Smart Agents in Industrial Cyber-Physical Systems. *Proceedings of the IEEE*, 104(6), pp.1086–1101.
- Leitão, P., Marik, V. & Vrba, P., 2013. Past, Present, and Future of Industrial Agent Applications. *IEEE Transactions on Industrial Informatics*, 9(4), pp.2360–2372.
- Leitão, P. & Restivo, F., 2006. ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, 57(2), pp.121–130.
- Lepuschitz, W., Zoitl, A., Vallée, M. & Merdan, M., 2011. Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1), pp.52–69.
- Lüder, A., Estévez, E., Hundt, L. & Marcos, M., 2010. Automatic transformation of logic models within engineering of embedded mechatronical units. *The International Journal of Advanced Manufacturing Technology*, 54(9–12), pp.1077–1089.
- Lüder, A., Estévez, E., Hundt, L. & Marcos, M., 2011. Automatic transformation of logic models within engineering of embedded mechatronical units. *International Journal of Advanced Manufacturing Technology*, 54(9–12), pp.1077–1089.
- Marcos, M., Estevez, E., Perez, F. & Van der Wal, E., 2009. XML exchange of control programs. *IEEE Industrial Electronics Magazine*, 3(4), pp.32–35.
- Marín, C.A., Mönch, L., Leitão, P., Vrba, P., Kazanskaia, D., Chepegin, V., Liu, L. & Mehandjiev, N., 2013. A Conceptual Architecture Based on Intelligent Services for

- Manufacturing Support Systems. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*. pp. 4749–4754.
- Merz, M., Frank, T. & 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)*. IEEE, pp. 863–868.
- Morenas, J. de las, Higuera, A.G. & Alonso, P.G., 2012. Product Driven Distributed control system for an experimental logistics centre. *International Journal of Innovative Computing, Informatics and Control*, 8(10), pp.7199–7216.
- Nouri, H., 2015. Development of a comprehensive model and BFO algorithm for a dynamic cellular manufacturing system. *Applied Mathematical Modelling*, 40(2), pp.1514–1531.
- Olsen, S., Wang, J., Ramirez-Serrano, A. & Brennan, R.W., 2005. Contingencies-based reconfiguration of distributed factory automation. *Robotics and Computer-Integrated Manufacturing*, 21(4), pp.379–390.
- Onori, M., Semere, D. & Lindberg, B., 2011. Evolvable systems: an approach to self-X production. *International Journal of Computer Integrated Manufacturing*, 24(5), pp.506–516.
- Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovc, N., Quilici, A., Rosenblum, D.S. & Wolf, A.L., 1999. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, 14(3), pp.54–62.
- Pang, C. & Vyatkin, V., 2010. IEC 61499 function block implementation of intelligent mechatronic component. *IEEE International Conference on Industrial Informatics (INDIN)*, pp.1124–1129.
- Ribeiro, L., Barata, J., Onori, M. & Hoos, J., 2015. Industrial Agents for the Fast Deployment of Evolvable Assembly Systems. In P. Leitão & S. Karnouskos, eds. *Industrial Agents*. Boston: Morgan Kaufmann, pp. 301–322.
- Rocha, A., Orío, G. Di, Barata, J., Antzoulatos, N., Castro, E., Scrimieri, D. & Ribeiro, L., 2014. An Agent Based Framework to Support Plug And Produce. In *12th IEEE International Conference on Industrial Informatics (INDIN), 2014*. pp. 504–510.
- Salehie, M. & Tahvildari, L., 2009. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2), pp.1–42.
- Schilit, B.N., Adams, N. & Want, R., 1994. Context-aware computing applications. *IEEE Workshop on Mobile Computing Systems and Applications*, pp.85–90.
- Schimmel, A. & Zoitl, A., 2011. Distributed online change for IEC 61499. In *IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, pp. 1–

7.

- Schmidt, D.C., 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(February), pp.25–31.
- Schütz, D., Obermeier, M. & Vogel-heuser, B., 2013. SysML-Based Approach for Automation Software Development – Explorative Usability Evaluation of the Provided Notation. In A. Marcus, ed. *Design, User Experience, and Usability. Web, Mobile, and Product Design*. Springer Berlin Heidelberg, pp. 568–574.
- Schütz, D., Wannagat, A., Legat, C. & Vogel-Heuser, B., 2013. Development of PLC-Based Software for Increasing the Dependability of Production Automation Systems. *IEEE Transactions on Industrial Informatics*, 9(4), pp.2397–2406.
- Science, N. & Council, T., 2016. ADVANCED MANUFACTURING : A Snapshot of Priority Technology Areas Across the Federal Government Subcommittee for Advanced Manufacturing. , (April).
- Selic, B., 2003. The pragmatics of model-driven development. *IEEE Software*, 20(5), pp.19–25.
- Shen, W., Wang, L. & Hao, Q., 2006. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 36(4), pp.563–577.
- Standards Board, I., 1990. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990IEEE Std 610.12-1990*, pp.1–84.
- Strasser, T. & Froschauer, R., 2012. Autonomous Application Recovery in Distributed Intelligent Automation and Control Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6), pp.1054–1070.
- Streit, A., Rösch, S. & Vogel-Heuser, B., 2014. Redeployment of Control Software during Runtime for Modular Automation Systems Taking Real-Time and Distributed I/O into Consideration. In *IEEE 19th Conference on Emerging Technologies Factory Automation (ETFA), 2014*. pp. 1–4.
- SysML, 2007. The SysML Specification, v 1.0. Available at: <http://www.sysml.org>.
- Thramboulidis, K., 2006. Design alternatives in the IEC 61499 function block model. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pp.1309–1316.
- Thramboulidis, K., 2013. IEC 61499 vs. 61131: A Comparison Based on Misperceptions. *arXiv*, 2013(August), pp.3–5.
- Thramboulidis, K., 2005. Model-Integrated Mechatronics—Toward a New Paradigm in the Development of Manufacturing Systems. *IEEE Transactions on Industrial Informatics*, 1(1), pp.54–61.

- Thramboulidis, K., 2010. The 3+1 SysML View-Model in Model Integrated Mechatronics. *Journal of Software Engineering and Applications*, 3(2), pp.109–118.
- Thramboulidis, K., 2011. Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation. *Journal of Software Engineering and Applications*, 4(4), pp.217–226.
- Thramboulidis, K., Perdikis, D. & Kantas, S., 2006. Model driven development of distributed control applications. *The International Journal of Advanced Manufacturing Technology*, 33(3–4), pp.233–242.
- Urban, T.L. & Chiang, W.C., 2016. Designing energy-efficient serial production lines: The unpaced synchronous line-balancing problem. *European Journal of Operational Research*, 248(3), pp.789–801.
- Vogel-Heuser, B. & Rösch, S., 2014. Integrated Modeling of Complex Production Automation Systems to Increase Dependability. In *Risk - A Multidisciplinary Introduction*. pp. 1–476.
- Vogel-Heuser, B., Schütz, D., Frank, T. & Legat, C., 2014. Model-driven engineering of Manufacturing Automation Software Projects - A SysML-based approach. *Mechatronics*, 24(7), pp.883–897.
- Vrba, P., Tichý, P., Mařík, V., Hall, K.H., Staron, R.J., Maturana, F.P. & Kadera, P., 2011. Rockwell Automation's Holonic and Multiagent Control Systems Compendium. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1), pp.14–30.
- Vyatkin, V., 2011. IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 7(4), pp.768–781.
- Vyatkin, V., Christensen, J.H. & Lastra, J.M., 2005. OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation. *IEEE Transactions on Industrial Informatics*, 1(1), pp.4–17.
- Vyatkin, V. & Hanisch, H.-M., 2009. Closed-Loop Modeling in Future Automation System Engineering and Validation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(1), pp.17–28.
- Van der Wal, E., 2009. PLCopen. *IEEE Industrial Electronics Magazine*, 3(4), p.25.
- Wang, L., Adamson, G., Holm, M. & Moore, P., 2012. A review of function blocks for process planning and control of manufacturing equipment. *Journal of Manufacturing Systems*, 31(3), pp.269–279.
- Wannagat, A. & Vogel-Heuser, B., 2008. Agent oriented software-development for networked embedded systems with real time and dependability requirements in the domain of automation. In *17th World Congress The International Federation*

- of Automatic Control*. Seoul, Korea, pp. 4144–4149.
- Wegdam, M., Almeida, J.P.A., Sinderen, M.J. van & Nieuwenhuis, L.J.M., 2003. Dynamic Reconfiguration for Middleware-Based Applications. *IEEE Transactions on Parallel and Distributed Systems*, (Fall 2003), pp.1–30.
- Wehrmeister, M.A., de Freitas, E.P., Binotto, A.P.D. & Pereira, C.E., 2014. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. *Mechatronics*, 24(7), pp.844–865.
- Wooldridge, M., 2009. *An Introduction to Multi-Agent Systems* J. W. & Sons, ed.,
- Yan, J. & Vyatkin, V., 2013. Extension of reconfigurability provisions in IEC 61499. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, pp. 1–7.
- Yang, C.W., Yan, J. & Vyatkin, V., 2013. Towards implementation of Plug-and-Play and distributed HMI for the FREEDM system with IEC 61499. *IECON Proceedings (Industrial Electronics Conference)*, pp.5347–5353.
- Zhang, J., Khalgui, M., Li, Z., Frey, G., Mosbahi, O. & Salah, H. Ben, 2015. Reconfigurable Coordination of Distributed Discrete Event Control Systems. In *IEEE Transactions on Control Systems Technology*. pp. 323–330.
- Zhou, B., Li, C. & Zhao, X., 2007. FIPA agent-based control system design for FMS. *The International Journal of Advanced Manufacturing Technology*, 31(9–10), pp.969–977.







## **GLOSSARY**

---



## **Glossary**

**ACC** - Agent Communication Channel

**ACL** - Agent Communication Language

**AML** - Automation ML

**AMS** - Agent Management System

**CAEX** - Computer Aided Engineering eXchange

**CA** - Controller Agents

**D&D** - Diagnosis&Decision

**DF** - Directory Facilitator

**FAF** - Flexible Automation Framework

**FAM** - Flexible Automation Middleware

**FAPS** - Flexible Automation Production System

**FB** - Function Block

**FIPA** - Foundation for Intelligent Physical Agents

**FSM** - Finite State Machine

**IE** - Internal Element

**JADE** - Java Agent DEvelopment Framework

**M2M** - Model to Model

**M2T** - Model to Text

**MAS** - Multi Agent Systems

**MC** - Mechatronic Component

**MCA** - Mechatronic Component Agents

**MDD** - Model-Driven Design

**MDE** - Model-Driven Engineering

**MM** - Middleware Manager

**QM** - Quality of Service Monitor

**QoS** - Quality of Service

**SCI** - Source Code Insertion

**SOA** - Service Oriented Architecture

**SUC** - System Unit Classes

**SysML** - Systems Modeling Language

**UML** - Unified Modelling Language

**XML** - eXtensible Markup Language