

A method for automation software design of mechatronic systems in manufacturing

P. Sangregorio*, A. L. Cologni*, A. Piccinini***,
A. Scarpellini*, F. Previdi*

* *Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione. Università degli Studi di Bergamo, Via Galvani 2, 24044 Dalmine (BG) (e-mail: paolo.sangregorio@unibg.it).*

** *Consorzio Intellimech*

Abstract: The fast globalization process of the last decade required companies to optimize internal processes. Additionally, customers requirements are becoming more and more specific, requiring strong customization. In the mechatronic field, the automation software development takes a wide percentage of the overall designing time, due to the lack of structured design processes (specially for small systems) and code reuse.

This paper aims to define a first structured approach for the automation software development of small manufacturing systems. The proposed solution is based on concepts like modularization and encapsulation, trying to build a one-to-one relationship between the physical mechatronic object and its control software module. Each module embeds the control software and other information related to the HMI, the documentation and the risk assessment.

The introduced method has been implemented on a real small manufacturing system (an industrial manipulator). The components that form the device have been identified and implemented as modules. The design phase becomes, then, a drag and drop composition of such modules, controlled by a state machine that describes the working cycle of the device.

© 2015, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: Automation, Flexible automation, Flexible manufacturing systems, Software engineering, Computer-aided manufacturing.

1. INTRODUCTION

In the last decades, competition has become the leading theme that manufacturing companies have faced. The process of globalization has meant that local and fragmented markets has been merged in a single market, in which companies of emerging economies are faced with those of mature economies. This process led to some undesired consequences: the reduction of the cost of the product (because of the increased supply) and, obviously, the reduction of profit margins. Those requirements have driven massive introduction of intelligent components into the machine design, and the integration of such devices requires more and more advanced technologies, see Terzic et al. (2008). On the other hand this process, combined with the strong technological evolution, allowed to increase the customer base and to decrease the production costs. In this evolving scenario, the companies that are able to reduce the development costs can obtain an advantage over the competitors.

Until the last decade, in the field of mechatronic systems the development was based on a design-by-discipline approach, see Harashima et al. (1996). In such approach, the design consists of a succession of activities. The design of an electromechanical system for instance, was often accomplished in three steps. First, the mechanical parts were designed and when the mechanical development was complete, the power and microelectronics were designed and finally, the control algorithm was planned and im-

plemented, see Shetty and Kolk (2010). This sequential approach is inefficient: each sequence locks some aspects that become additional constraints to the next sequence; another drawback of this approach, is that it is time consuming since each team can begin working only when the previous has finished. For these reasons, in the last years, this development scheme has been replaced with the SE (Systems Engineering) approach, see Rouse (2003), Haskins et al. (2006), Shishko (1995), DoD (2001) and Pysteer and Olwell (2013). This approach defines the organization and the development of complex mechatronic systems, such as the development of a spacecraft system (Eickhoff and Roeser (2009)), but doesn't define a guideline that can be applied to small and customizable systems.

This work aims to define a first software development solution in the context of small and customizable manufacturing systems. The proposed solution is based on a work-flow (see Figure 2) that includes the following steps:

- definition of the product specs;
- definition of the mechatronic modules;
- development of the control system state machine;
- automatic generation of the HMI (Human Machine Interface) and the documentation.

The basic component of the solution described in this paper, is the mechatronic object. Several authors exploited this concept, like Thramboulidis (2008) where the MTC

(Mechatronic Components and Connectors) concept is introduced for building complete applications starting from basic elements.

The proposed approach has been tested and verified on a real mechatronic system, see Previdi et al. (2010) and Previdi et al. (2012). This device has some characteristics that make it usable as a test bench. The system is basically made of a motor that helps the operator in lifting heavy weights. In the possible configurations of the product (more than one hundred) the parts of the system are different or connected in different ways; that makes this device perfect for applying the proposed method. Instead of redesign the whole system from scratch each time a new configuration is needed, or starting from a different version and adapting it, this method allows reuse of software by modules. Then the automation software is built as a composition of modules.

The paper is organized as follows. In Section 2, the landscape of applicability of the methodology is introduced, and the approach presented. In Section 3, the described method is implemented and tested on a real case (an industrial manipulator). This section highlights the tools used for applying the presented work-flow, and provides examples of the results. Sections 4 is devoted to the final assessments and the future developments.

2. THE PROPOSED APPROACH

When considering families of manufacturing systems that differ only for some specific parts, but sharing a common base, it is easy to imagine that the most of the automation software that controls these machines is common. Additionally, most of these machines uses mechatronic parts that are always the same even if interconnected in different ways or with different configurations. This makes evident that writing the automation software from scratch each time is a waste of time and resources. The approach, presented in this section, aims at separating the software in modules to match the mechatronic components that are present on the machine. This introduces a way to assemble and reuse those modules with the purpose of reducing the time and effort needed for the development of the control software. The concept of module replicates the separation between mechatronic modules in software. The "software" modules, in fact, includes the automation software of the relative hardware module, leveraging concepts like incapsulation and information hiding, typical of high level languages. This can be applied in such context by the mean of providing a defined interface for invocation and communication with the module, regardless of its internal implementation. In such a way, the development of the automation software relies on already built and tested blocks making the assembly of the machine software an easy and faster task, compared to the monolithic development. This is a key point of modularization. Having the ability of selecting pieces of software that are already validated and secured moves the effort of verification and testing from the whole code to only the integration between blocks, which is basically the working cycle of the machine that commands those modules.

In the proposed solution, in addition to the automation software that implements the control and the logic needed

for managing a mechatronic object, a module has various metadata associated:

- versioning information
- description and documentation for automatically compose the documentation
- interfaces for reading data and invoking actions
- configurable properties for changing its behavior (i.e. different configurations of the hardware module that need to be considered in the software)
- risks associated to the specific module for generating risk documents
- runtime configurable parameters depending on the module for the automatic generation of the configuration menus

The illustrated approach describes how automation software can be easily built by composition of such modules, leveraging modularity and encapsulation. Another important outcome of the proposed architecture is the possibility to store the software modules in a versioned database that can be accessed each time a module is needed in the development. This solution guarantees:

- the usage of the latest verified and validated version for each module (instead of relying on copy and paste techniques widely used in the industry, that are error-prone)
- high traceability of issues and bugs in control code. Each time a issue is discovered within a specific block, it is easy to find all the shipped devices that run that bug thanks to the versioning of all the used modules, and release an update for them.
- the increase of hierarchical levels in software development: the software module developer is different from who develops the machine working cycle. This allows developers to concentrate on smaller parts of the whole machine increasing the reliability of the developed code.

The designed approach consists of the steps represented in figure 1.

(1) Selection of modules that compose the mechatronic system

In this phase, the modules that are present in the hardware configuration of the target manipulator are identified. This can be easily done by mapping software modules one-to-one with mechatronic objects. If this correspondence is in place, the selection of modules can also be performed automatically depending on the mechatronic objects selected by mechanical or electrical engineers during the design of the hardware. The automation software in fact, will be a composition of all the software of each single block. The selection is made among the available modules in the database. The interfaces of the selected modules

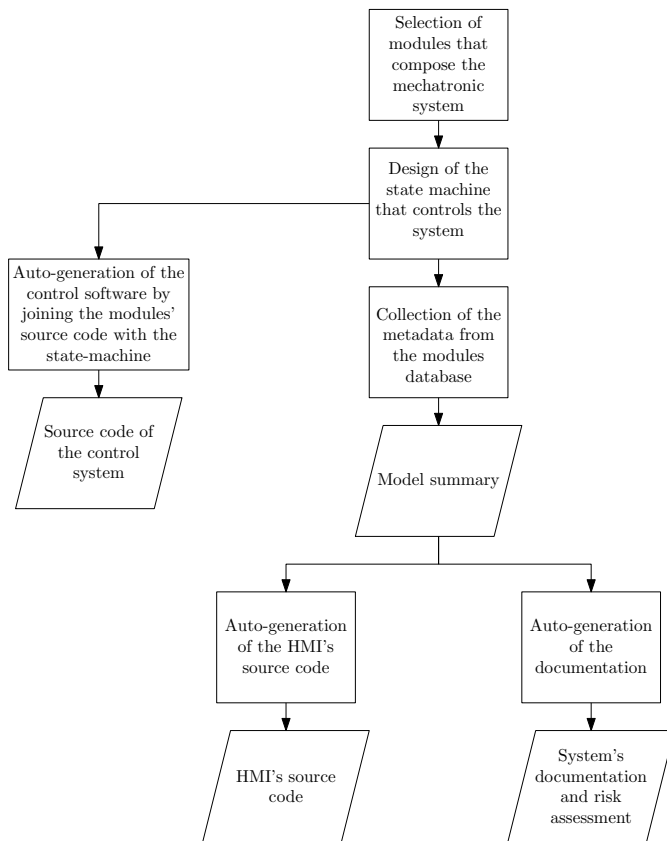


Fig. 1. The proposed approach

are then connected in a graphical representation to reproduce the configuration on the hardware and their properties are set accordingly. A property is an internal variable that can assume only some specified values, needed for include all the possible variations of a single module. For example, there can be different configurations of the same module, one with additional safety sensors and another without. The presence or not of the safety sensors is a property, that can be set or not, changing the behavior of the module, and of the automation software as well. Once again, all these changes in the module behavior are implemented inside the module, and can be enabled or disabled just by setting a property on it.

(2) Design of the state machine that controls the system

Once the modules are in place, the next phase is the design of the machine cycle. This step in the proposed architecture is built using a state machine, that operates directly on the modules by invoking actions, or reading data from them. This state machine is represented in a graphical environment, and the conditions for moving from one state to another are depending on the inputs coming from the blocks. Each state can have entry actions (executed the first time the state is entered), during actions (executed at automation cycle) and exit actions (executed when exit from the state).

(3) Auto-generation of the control software by joining the modules' source code with the

state-machine

Once the working cycle has been defined, the generation of the automation software takes place. This step is built using code generation tools that, starting from the modeled machine cycle and the software of the included blocks, generates the control software needed for running the manufacturing system. This generated software is then joined with a firmware layer that contains all the low level functions that can be called by the modules for communicating with the hardware.

(4) Collection of the metadata from the modules database

The modules database is read looking for the metadata of the used modules. This metadata contain all the information needed for building documentation, risk assesment and the HMI source code.

(5) Auto-generation of the HMI's source code

Depending on the modules that are available on the manipulator, the user menus are automatically generated with the information provided by the module definition. These menus contain all the visualization and configuration parameters corresponding to the included modules.

(6) Auto-generation of the documentation

The last step of the proposed process is the generation of the documentation. As stated before, each module contains also information related to the documentation and risk assessment, providing a way for automatically compose the base structure and content of the documentation, again in a modular way. This allows the development team to focus on specific additional information that need to be inserted in the documentation, without worrying about all the warnings and instructions related to the modules. Additionally, each time a module's behavior is changed for fixing issues, or just for improving its performances, an update of the documentation can be easily released too, just by updating the module specific information in the database, and rebuilding the whole documentation.

Figure 2 represents the composition of the automation software starting from the database of software modules.

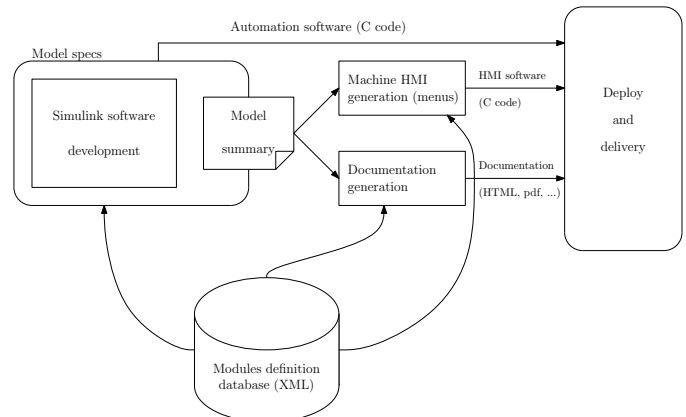


Fig. 2. Composition of the automation software

3. APPLICATION OF THE DESCRIBED APPROACH ON AN INDUSTRIAL MANIPULATOR

As an example of the proposed method, a manufacturing system has been considered. The system is an industrial manipulator (see Figure 3) that is composed by a vertical body, a brushed motor on the top, an arm and a rope linked to the motor at one end, and to an end effector at the other end. The end effector can be of various types, depending on the kind of load it has to lift. This device allows workers to lift an heavy load without effort by hanging it to the end effector, compensating the weight of the lifted object with the motor. The system is available with a wide variety of configurations of the end effector, ranging from electronic grippers to pneumatic vacuum pumps. Some end effectors allow also rotation of the load, or provide additional features compared to the base version, like balancing the load for keeping it in the correct position while lifting it.

As it can be guessed from the description, the majority of the software of such a manufacturing system is common regardless of the specific configuration of the end effector making this device perfect for the proposed approach. Each time a new machine with a different end effector is required, a considerable part of the software can be reused, adding just the part of automation software that controls the particular tool.

The end effector mounted on the considered example is composed of a gripper, a button for closing the gripper and one for opening it, an handle for moving the load vertically and a load balancer that moves back and forth for keeping the load in vertical position.

The identified modules on this specific manipulator are:

- *handle*: it reads the force applied by the operator. It is basically an input module so the block has only one output interface, which is the read value (the input of the user becomes the output of the block).
- *load balancer*: it is basically composed by a gear connected to a rack. When the gear rotates the rack moves back or forth. There are two versions of this load balancer, one with two end switches for detecting if the gear is hitting the end in one or in the other direction, and another with a potentiometer that gives the absolute position of the gear relative to the rack. Depending on the module mounted on the hardware, an option of this software module need to be set. The software block has then two or one outputs (two booleans that refers to the end switches, or just one real that gives the gear position) depending on the configuration, and one input that represents the command to the motor. The load balancer used in this example has two end switches.
- *gripper*: it has two end switches for determining if it is fully open or fully closed. The block, then, has an input for providing the command to the gripper, and two outputs for reading the value of the end switches.

- *input button*: this module is present twice in the given manipulator. This is only a button that the operator uses as an input device, so the block has only one output that gives the status of the button.
- *load cell*: it is a sensor for reading the load attached to the manipulator. The software block has only one real output which gives the value of the attached load.
- *display*: it is the main HMI of the manipulator. Its software block has basically just an input interface that allows to set the displayed text.
- *lamp*: this module represents an output lamp that can be turned on or off, by changing the value of its input.
- *motor*: it compensates the load. This module has three inputs, the torque needed for compensating the weight of the lifted object, and two maximum speeds for lifting and dropping actions, for safety reasons.
- *load cell*: this module is used for measuring the load of the lifted object. It has basically just one output, the mesured weight.

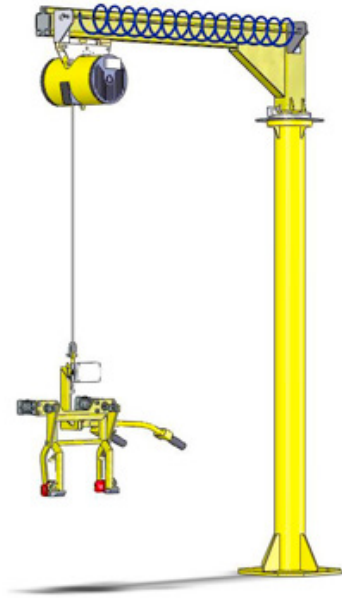


Fig. 3. The industrial manipulator

For applying the proposed method, all these modules are described using eXtensible Markup Language (XML) with a scheme defined to contain all the required metadata about versioning, interfaces, documentation and parameters. All these information will be used for generating the documentation and HMI as described later in this section.

Once the modules definition is ready, the corresponding software module is developed. The selected environment for this application is Matlab Simulink due to its natural modularity given by its block structured design and its capability of generating C source code that can be run on the manipulator ECU. Each software module is implemented to match interfaces and definition of its XML description and stored in a Simulink library. The working cycle of the machine is implemented using a Finite State Machine

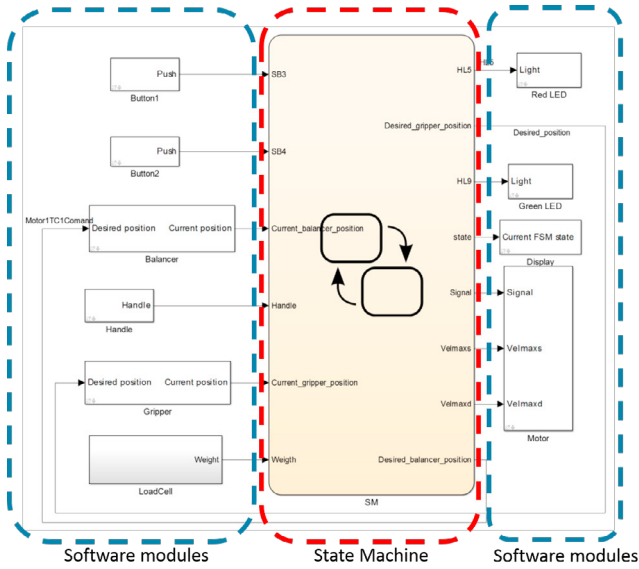


Fig. 4. The automation software

that interacts with modules by the means of input and output signals that drive the state transitions within the FSM and command the hardware modules depending on the internal FSM state. The FSM is implemented using Matlab Stateflow which allows static and run-time checks for cyclic problems and state inconsistencies for validating the developed working cycle. The software of the given example is presented in Figure 4. Each block provides input or takes outputs from the state machine. Each block, internally implements the control logic of that mechatronic part. Figure 5 represents the internal part of the *Balancer* block as an example. The block reads two end switches from the hardware and, given the desired position, commands the motor updating the current position as soon as the desired position is reached.

The communication with the low level hardware is accomplished with *Input read* and *Output write* blocks that allows to link the signal from and to such blocks to the input/output pins of the control unit on the hardware. Similar blocks are provided also for communication with the CAN bus. This mapping is performed within the configuration of the block that has been realized as shown in Figure 6. All the software modules represented in Figure 4 follow the same structure as the *Balancer* module just described. The state machine that implements the working cycle of the manipulator is presented in Figure 7. Each transition depends on some inputs that are coming from the modules while actions that need to be accomplished on modules are written within the states.

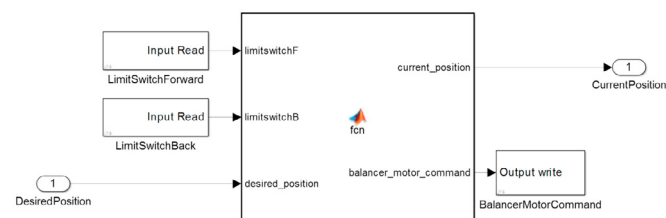


Fig. 5. Internal view of the balancer module

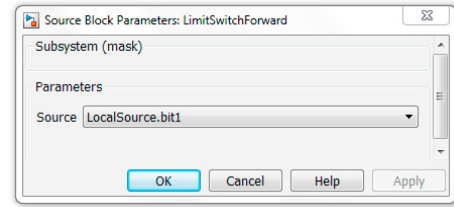


Fig. 6. The mapping between software signals and hardware pins

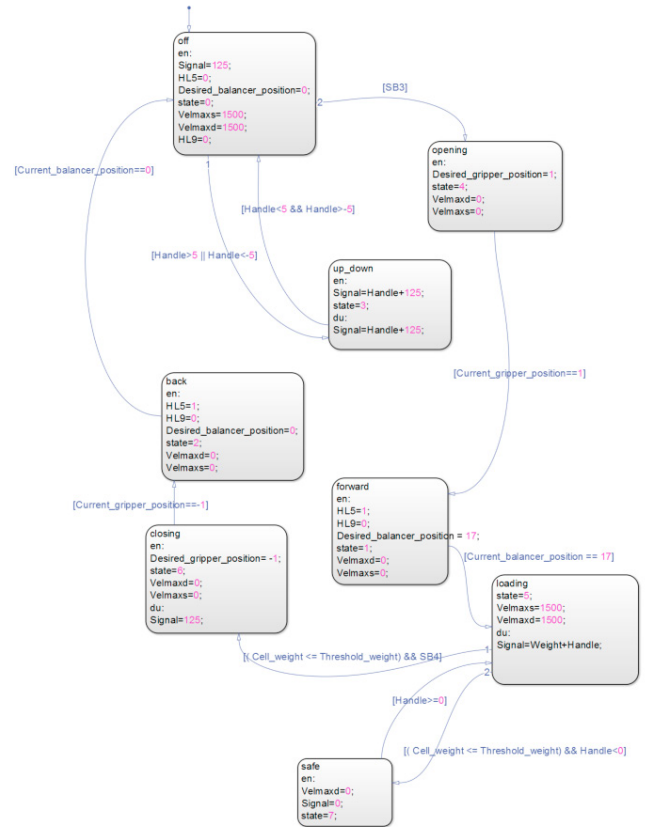


Fig. 7. The state machine that implements the working cycle

All the *Input read* and *Output read* blocks rely on some low level functions that are implemented in the *Base firmware*, which is the common part of base functions that are common among the various versions of this manipulator. More in detail, it contains all the communication functions needed by software modules, the main loop that runs the state machine, and the HMI manager that takes care of controlling the navigation in HMI menus.

The content of this HMI menu is automatically generated by a tool developed for this application. For automatically generating the HMI, a list of the modules used within the control software is needed. An XML file containing a list of the used modules and a description of the connection between them in the Simulink model is exported using a Matlab script. This file, called *model summary* is used as input by the HMI generator. The generator, reads all the modules contained in the model, and the corresponding XML definition files in the repository for gathering information about the menu items needed for the used modules.

This tool generates the C code needed for controlling the menu.

The same process is followed for generating the software documentation and user manual. Depending on the modules, the corresponding documentation is loaded from the repository and joined in a single document. Thanks to a multilanguage setup of the XML format of the modules, the documentation can be generated in various languages without any effort.

4. CONCLUSIONS AND FUTURE DEVELOPMENT

In this paper an architecture for the development of small modular and customizable manufacturing systems is presented. The proposed work-flow is a structured approach that aims at building the automation software by composing prebuilt modules leveraging concepts like encapsulation and modularity. This approach improves the time to market when applied to products that can have a lot of configurations, assembled in many different ways. One of the main outcomes of using the discussed method is the overall software reliability: by using already validated modules, the testing focus is moved on the integration of those modules. Another outcome is the automatic generation of documentation and HMI that are tasks that are easily automated once the desired result is defined. The main drawback of this approach is that even if the modular paradigm for automation software can be achieved as described with almost every mechatronic system, the generation of the documentation and HMI requires a specific knowledge of the application because the generated code is strictly linked with the type of the device. This requires the final steps of the work-flow to be customized depending on the company needs, but once those steps has been customized, the method presents benefits in terms of speed and reliability of the automation software development.

ACKNOWLEDGEMENTS

This research activity has been carried out in the context of the research project *ADAPTIVE, Approccio modulare ed adattativo alla Fabbrica Digitale* funded by Ministero dell'Istruzione, Università e Ricerca.

REFERENCES

- DoD (2001). System engineering fundamentals". *Defense Acquisition University Press*.
- Eickhoff, J. and Roeser, H.P. (2009). *Simulating spacecraft systems*, volume 1. Springer.
- Harashima, F., Tomizuka, M., and Fukuda, T. (1996). Mechatronics-'what is it, why, and how?' an editorial. *IEEE/ASME Transactions on Mechatronics*, 1(1), 1–4.
- Haskins, C., Forsberg, K., Krueger, M., Walden, D., and Hamelin, D. (2006). *Systems engineering handbook*. In *INCOSE*,.
- Previdi, F., Fico, F., Belloli, D., Savaresi, S., Pesenti, I., and Spelta, C. (2010). Virtual reference feedback tuning (vrft) of velocity controller in self-balancing industrial manual manipulators. In *American Control Conference (ACC)*, 2010, 1956–1961. IEEE.
- Previdi, F., Fico, F., Savaresi, S.M., Belloli, D., and Pesenti, I. (2012). Direct design of a velocity controller and load disturbance estimation for a self-balancing industrial manual manipulator. *Mechatronics*, 22(8), 1177–1186.
- Pysteer, A. and Olwell, D. (2013). *The Guide to the Systems Engineering Body of Knowledge v. 1.1*.
- Rouse, W.B. (2003). Engineering complex systems: Implications for research in systems engineering. *Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *IEEE Transactions on*, 33(2), 154–156.
- Shetty, D. and Kolk, R. (2010). *Mechatronics System Design, SI Version*. Cengage Learning.
- Shishko, R. (1995). *NASA System Engineering Handbook*. National Aeronautics and Space Administration.
- Terzic, I., Zoitl, A., Favre, B., and Strasser, T. (2008). A survey of distributed intelligence in automation in european industry, research and market. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 221–228. IEEE.
- Thramboulidis, K. (2008). Challenges in the development of mechatronic systems: The mechatronic component. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 624–631. IEEE.