



UNIVERSITY OF BERGAMO

School of Doctoral Studies

Doctoral Degree in Engineering and Applied Sciences

XXIX Cycle

**Integrated methodologies of software development and remote
maintenance for semi-automated machines**

Advisor

Chiar.mo Prof. Fabio Previdi

Doctoral Thesis

Paolo Sangregorio

Student ID 1006895

Academic year 2015/16

Abstract

Day after day, vendors of manufacturing machineries are challenged with the need of flexibility and reconfigurability of their products. Being able to rapidly provide various variants and configurations of the products in response to customers requests is an important asset for every company in the nowadays competition. Nevertheless, most of the automation software for special machines is still built in a monolithic way which does not support rapid reconfigurability to reflect hardware modifications. And once the machine is delivered to the customer site, the maintenance process begins. Maintenance costs are increasing with the globalized market due to travel costs for sending service personnel to the customer plants. In some cases, the issue would have been easily solved if only the technician had been able to be on the site. This makes remote diagnosis, debugging and repair an obvious advantage.

This thesis presents two approaches to solve the aforementioned issues. A modular and reconfigurable approach to software development for the automation to support rapid reconfigurations of the software, and a remote maintenance methodology that uses wireless and mobile technology to record and transmit video and machine operational parameters together for remote viewing and analysis.

Contents

1	Introduction	1
1.1	The need of an integrated methodology	1
1.2	Contributions	3
1.3	Thesis structure	4
2	State of the art	7
2.1	Modularization and modeling	8
2.1.1	Modularization attempts	11
2.1.2	Software product lines	13
2.1.3	Model based design approach	16
2.2	Remote maintenance	29
2.2.1	E-Technologies for maintenance improvements	29
2.2.2	E-maintenance	31
2.2.3	Web technologies as enabling factors for re- mote maintenance	32

3	Integrated methodology	37
3.1	Automation software development method for semi-automated machines	38
3.1.1	The concept of module	39
3.1.2	The process	49
3.2	Development of a remote maintenance system for semi-automated machines	58
3.2.1	System architecture	60
3.2.2	Remote assessment process	63
3.2.3	Smart device app	68
3.2.4	Headquarters server application	69
4	Implementation and use case	73
4.1	Software development	75
4.1.1	Definition and selection of modules	76
4.1.2	Design of the control software	81
4.1.3	Exporting of a model of the system	82
4.1.4	Automatic generation of HMI interface	84
4.1.5	Automatic generation of documentation	86
4.2	Remote maintenance	87
4.2.1	The manipulator architecture	88
4.2.2	The gateway	89
4.2.3	Smartphone app	93

4.2.4	Windows app	98
4.2.5	Synchronization	103
5	Conclusions and future work	109
5.1	Limits and strenghts	111
5.1.1	Automation software development method: limits and strenghts	111
5.1.2	Remote maintenance solution for semi-automated machines: limits and strenghts	112

List of Figures

2.1	Sequential approach	9
2.2	An example of feature model	15
2.3	SysML diagram taxonomy as presented on the official website www.omg.sysml.org	25
3.1	Helicopter view of the proposed methodology	40
3.2	The composition of machine automation software	56
3.3	The process for generation of the machine software	56
3.4	The architecture of the remote maintenance system	61
3.5	The data acquisition process	64
3.6	The firmware upgrade process	67
3.7	The headquarters server application architecture	70
4.1	The manipulator used in the use case	75
4.2	The working cycle designed in Stateflow	83

4.3	The connection between software modules and the state machine	84
4.4	The internal implementation of the balancer module	85
4.5	An example of the generated documentation for menu navigation	87
4.6	Message protocol structure	93
4.7	The data packet structure	94
4.8	The architecture of the smart device application . .	95
4.9	The developed Windows application	99
4.10	The sections of the Windows application	100
4.11	The MATLAB exported file	102
4.12	The phases of a recording	105
4.13	Transmission scheme	106
4.14	Test scheme	108

Chapter 1

Introduction

1.1 The need of an integrated methodology

Every day, producers of manufacturing systems are challenged with the need of flexibility and reconfigurability of their products. The ability of rapidly provide various configurations of products in response to customers demand is an important asset in the ever growing competition of nowadays. Competition is about speed and costs, which means that being able to respond fast and contain costs is crucial to gain new customers. Nevertheless, most of the automation software for special machines is developed in a monolithic way which does not support rapid reconfigurability to

reflect hardware changes. Several solutions have been proposed for supporting rapid reconfiguration of production lines, but the field of custom machines is still fertile about improvements in this area. Due to the increasing of globalization, machineries are shipped all over the world. This growth of exportation of manufacturing systems increased costs of maintenance in case of failure or malfunctioning of the machine, which can become really expensive due to the need of sending a technician to the customer site. And many times service personnel is moving just for issues that can be solved with simple steps if only the technician could have seen it directly. Considering that each intervention is just a cost for the company, vendors need to find a way for managing the maintenance efficiently, even when machines are far away. Additionally, if the company is able to offer rapid maintenance because it can leverage remote support techniques, it can guarantee short downtimes in case of failure. It is evident that such short times of recovery could be achieved only if specialized personnel is available on the customer site, or if the technicians from the vendor site could somehow resolve the issue remotely.

1.2 Contributions

This thesis aims at exploring and proposing solutions for the two main challenges identified above. The first challenge is the development of a support system for the software generation and provisioning of a custom machine [43]. This thesis presents an approach based on the concept of module, which can be used by manufacturers of these machines to support the automatic generation of software depending on the components which are loaded on the custom machine. Additionally, other artifacts related to the software are automatically generated, allowing developers to concentrate on the integration of the various components, more than the development of the software for the components itself. The second challenge is to provide remote maintenance techniques for debugging issues remotely for semi-automated machines [44]. If on a fully automated system the machine log is most of the times enough to understand what conditions are triggering an issue, in the case of semi-automated machines the human interaction is valuable. It's sometime hard to understand what the operator of the machine is doing, or how he is interacting with the system when the wrong behavior manifests. This thesis provides a solution based on everyday mobile internet devices, which

tries to augmentate the information gathered from the machine with a audio/video stream acquired by a device to record the interaction that the user is having with the machine. This allows technicians on the vendor service center to audit the data having an idea of what the user was doing when the data was acquired. The work described in this thesis, has been carried out in the context of two research projects: the first is the ADAPTIVE project promoted by “Cluster Tecnologico Nazionale Fabbrica Intelligente” and funded by MIUR. The ADAPTIVE project aims at developing technologies and solutions to enhance the ability of modern factories to be flexible and efficient, appropriately responding to unpredictable changes in market requests. The second project is “Touchplant”, a project funded by Regione Lombardia and Fondazione Cariplo. This has the objective of “Developing innovative maintenance techniques for industrial machineries and plants, leveraging modern communication technologies and mobile terminals”.

1.3 Thesis structure

This thesis is structured as follows: Chapter 2 explores the state of the art for the fields of software development in manufacturing

and remote maintenance. The most meaningful research contributions on these areas are highlighted and described, to provide an idea of the current status of the research. Chapter 3 describes the methodology for the proposed solution, highlighting the structure and the expected outputs. Once the general approach is defined, Chapter 4 describes the application of the proposed solution on a real machine, with details about the implementation and the achieved results. To complete the thesis, Chapter 5 draws conclusions and highlights strengths and areas of improvement for the discussed approaches.

Chapter 2

State of the art

Producers of manufacturing machines and lines are faced day by day with the recurring problem of supporting a wide range of variants of their products to fulfill the ever increasing requirement of customization of their customers. Even if this thesis does not deal with customization directly, the modularization techniques described are one of the enabling factors of the customization. Additionally, the ability to answer in short time to market shifts and being able to reconfigure and redesign productive processes accordingly is a crucial asset in the globalized competition. This competition is the starting point of the current work, which tries to address this issue leveraging two main concepts:

- Modularization and automatic generation of automation soft-

ware and related artifacts

- Integrated remote maintenance systems

2.1 Modularization and modeling

Modularization and automatic generation of automation software and related artifacts increases development speed decreasing development costs, which means being able to respond fast and contain costs, crucial for gaining new customers. It introduces flexibility in the development and production process. Flexibility is a keyword in this context, and it allows to rapidly react to changes, whether predicted or unpredicted. First instances of flexibility in production were introduced by Toyota, as reported in [58] by Womack, Jones and Roos that revealed Toyota's lean manufacturing system. In that system, instead of relying on linear product lines, a paradigm involving self-coordinating work teams was the core of the production, while tools and procedures were put in place to support daily human work. Information sharing among the coworkers was the key point.

Despite this first introduction of lean manufacturing concepts, usual development in many companies is still carried out with a sequential approach, represented in Figure 2.1.

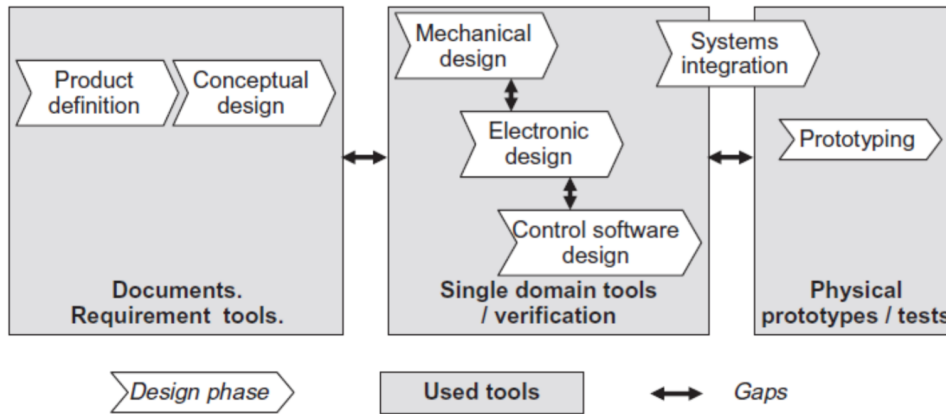


Figure 2.1: Sequential approach

This approach is mainly due to the specialization of engineers that work on product design and development. In a first instance, accountants or management defines the specifications of the product to be built, then a first high level architecture is designed. Once this is ready, mechanical engineers focus on materials and mechanical design, electrical engineers focus on control boards implementation, wiring and connection with electromechanical components, while control software designers concentrate on software implementation.

All these activities are often carried out in parallel but the lack of interaction between the various areas of knowledge is the main drawback of this approach. Every choice made by a team has impact on the others and thus communication is crucial. Addi-

tionally, making a design change late in the process can have a huge impact on other designs, because other teams have to implement that same change in their domain and be sure to not forget to address any issue related to that modification.

[4] highlights the main challenges of mechatronic design:

- **Sharing of design documents** Due to the high degree of interconnection between domains typical of mechatronic design, a complex system is often separated in smaller subsystems. These subsystems will be integrated into one single design later in the process, then communication and clear information flow between different working groups is critical to achieve a perfect integration.
- **Multi-discipline modeling** Modeling of mechatronic systems involves multiple disciplines, and thus, quitting a single-domain approach is crucial to favor knowledge sharing between designers and improve the overall design. This because every specialized designer can take decisions and make choices knowing what can be better for the other domains, facilitating their work and preventing changes late in the design process.
- **Early testing and verification** Early testing and verifica-

tion is primary to be able to verify the design in every stage of the process, before proceeding to the next phases. Frequent verification is another technique to prevent late changes.

- **Support in control software design** Usage of modern computer aided control system design tools such as Matlab or dSPACE [8] allow to develop the control software as block diagrams, providing a clearer and immediate representation of the control software, reducing development errors and bugs.

In the following sections, state of the art techniques to address these challenges are presented.

2.1.1 Modularization attempts

With the purpose of promoting modularization and code reuse in modern control software design for manufacturing systems the International Electrotechnical Commission (IEC) created the IEC 61499 [22] standard, that defines an open architecture to promote modularization and code reuse in modern control software design for the design of distributed control applications. The architecture of IEC 61499 is meant to be independent from the programming language trying to favour the reuse of software modules over multiple platforms. IEC 61499 has the core concept of Function Block,

inherited by a previous standard, defined in 1993 and published by IEC with the name of 61131-3 [19]. This concept of function block is basically a software structure that defines *Event inputs* and *Event outputs* for interacting with other function blocks, *Data inputs* and *Data outputs* for exchanging data with other function blocks, *Execution control chart* (ECC) to manage the internal state of the block depending on incoming events, and finally *Algorithms* that can be invoked by ECC to execute actions and update internal state. Every block implements its own logic, and thus a control application is obtained by an interconnection of those blocks, resulting in a network of function blocks [34]. The main difference between this standard and the previous IEC 61131-3 is the event-driven nature of the function block, which gets executed only upon receipt of an event. A review of the standard was needed because the previous version did not address the new requirements of today's complex industrial systems, missing key concepts like portability and reusability despite its wide adoption in the past [50]. Although IEC 61499 has been designed to overcome these lacks, it has scarcely been adopted by the industry, probably because of the lack of development tools that support this standard combined with the low interest by automation controllers manufacturers to support it.

2.1.2 Software product lines

Software product lines engineering [6] was introduced in 80's to increase economy of scale in products that had multiple versions or customization of components. The concept is similar to the product line concept typical of factories: given a pool of pre-designed components and customizations, a product line can realize multiple versions of the product by assembling those components. Most versions of products that come from a single factory in fact have most of the components in common, and only customizations or optionals are changing between them. The same concept is replicated in software development using *Software product lines* (SPL). [52] defines this concept: "A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way".

The purpose of this methodology is to reduce the time of development of different versions of software, creating variation points in the software to be able to choose what software component load depending on the hardware version of the product this software is installed on. SPL are based on four concepts:

- **Software assets** These are: (i) the main software with vari-

ation points and (ii) all the possible alternatives of software that can be loaded at those points to support specific alternatives of the product.

- **Feature model** The feature model [45], [3] is a tree representation of all the possible alternatives that can be achieved with the product line. It defines which components are optionals and which not, which alternatives are available for every component and whether a component requires other to be present. An example of a feature model is represented in Figure 2.2. In that model, a Manipulator is made up of a Motor, which can be *either* of Type A or Type B, a gripper which can be Mechanical or Pneumatic (or both), and a Balancer, which is not a mandatory component (note the empty circle in the connection line).
- **Production mechanism and process** The toolchain for composing and configuring products from the assets inputs. Product decisions are used during production to determine which software assets are needed and how to configure the variation points within those assets.
- **Software product outputs** The results of the software product line. It is represented by the collection of all the

products that can be produced by the product line. The scope of the product line is determined by the set of software product outputs that can be produced from the software assets and feature model.

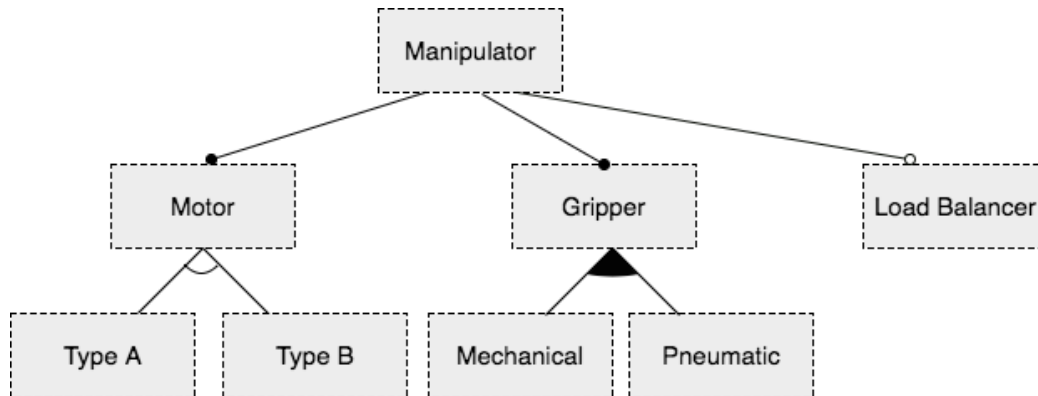


Figure 2.2: An example of feature model

The feature model describes all the alternative products that can be achieved using the software product line. Each feature of the tree can be mandatory, optional, or alternative. A feature is considered mandatory if it has to be present when the parent feature is present: in most cases, a mandatory feature is parent of many alternatives that can be chosen. This specifies that one of the alternatives can be selected, and that at least one have to be chosen. Optional features instead can be present or not present, while with alternative features exactly one feature must be selected when the father feature is selected. Representing all

the possible alternatives using this schema has the great advantage of removing ambiguities. Additionally, having a formal representation of the model, allows to validate production requests against the model to understand if a request is valid or not. There could be features that are conflicting, and thus the assembly of the software artifacts could not be possible. All those cases should be addressed by the validation of the request against the feature model.

An interesting comparison between a company that implemented Software Product Lines and one that did not is presented in [28]. The main benefit of Software Product Lines is that the production can usually scale to orders of magnitude more compared to the traditional software engineering techniques when considering wide ranges of product variants. Although this approach seems promising, its implementation requires the feature model to be well known from the very beginning to be able to design and implement all the variation points within the software.

2.1.3 Model based design approach

Model driven engineering is a software development methodology for systems engineering which exploits domain models rather than pure computing or algorithmic concepts. It has been promoted as

an approach to raise the abstraction level and better handling the complexity of modern industrial systems. The definition of Systems Engineering as provided by the International Council of System Engineering (INCOSE) is as follows:

Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems.

Systems engineering is related to:

- Definition and gathering of requirements
- Design
- Development
- System testing

Systems engineering then covers the whole development process, from the requirements to the testing, taking into account technical and economical aspects. Systems Engineering is a broad discipline which does not have a specific modeling language. The overall objective of this discipline is to reach the final objective (the working developed system) with a perfect and ideal integration of all the various aspects (from integration of technical components to tracing of requirements into specific subsystems).

The main phases of Systems Engineering as defined by INCOSE are [21]:

1. **State the problem** The problem statement starts with a description of the top-level functions that the system must perform: this might be in the form of a mission statement, a concept of operations or a description of the deficiency that must be ameliorated
2. **Investigate Alternatives** Alternative designs are created and are evaluated based on performance, schedule, cost and risk figures of merit. No design is likely to be best on all figures of merit, so multicriteria decision-aiding techniques should be used to reveal the preferred alternatives.
3. **Model the system** Models will be developed for most alternative designs. The model for the preferred alternative will be expanded and used to help manage the system throughout its entire life cycle. Many types of system models are used, such as physical analogs, analytic equations, state machines, block diagrams, functional flow diagrams, object-oriented models, computer simulations and mental models.
4. **Integrate** Systems, businesses and people must be integrated so that they interact with one another. Integration means

bringing things together so they work as a whole. Interfaces between subsystems must be designed. Subsystems should be defined to minimize the amount of information to be exchanged between them.

5. **Launch the system** Launching the system means running the system and producing outputs. In a manufacturing environment this might mean buying commercial off the shelf hardware or software, or it might mean actually making things. Launching the system means allowing the system do what it was intended to do.
6. **Assess performance** Figures of merit, technical performance measures and metrics are all used to assess performance. Figures of merit are used to quantify requirements in the tradeoff studies. They usually focus on the product. Technical performance measures are used to mitigate risk during design and manufacturing.
7. **Re-evaluate** Re-evaluate is arguably the most important of these functions. For a century, engineers have used feedback to help control systems and improve performance. It is one of the most fundamental engineering tools. Re-evaluation should be a continual process with many parallel loops.

8. **Variations** Like all processes, the Systems Engineering process at any company should be documented, measurable, stable, of low variability, used the same way by all, adaptive, and tailor-able.

System Engineering requires then a collaboration of experts from multiple domains, each of them having his own tools and procedures. To achieve a successful result, all these dependencies in domains should ideally be linked together and all the required information ubiquitously shared among the team. Every domain though has its own tools, and linking of information and components over multiple tools is still a complex issue. That's the main reason why languages like AutomationML have been developed in the last years, exactly with the purpose of facilitating exporting information from one tool and importing into another.

In the best of its shape, model based design aims at creating a multi-layer model which describes the system under various aspects. The top-level representation describes the most generic features or components of the model, but deep diving into each component will expand its content over multi disciplinary domains. Ideally, this single model is used for defining requirements, then extended linking these requirements to every component that implements them. The model is integrated with a description of the

composition of the system from an architectural point of view, then expanded again for including electrical specification and details about cabling, software components, verification and validation. Various attempts have been performed to achieve automatic software generation [10] starting from this model.

In the following sections, some of the most common model driven engineering languages are presented.

SysML

UML is the de-facto standard language for modeling software requirements. From the success of UML a language for engineering applications called SysML [47] has been derived. It supports design phases such as requirements collection and formalization of specifications. Hirsch [17] designed a way for linking function block technology with SysML [16]. SysML is suitable for the top-down design and requirement engineering, however it is less efficient when it comes to the deployment of distributed embedded targets. Additionally, legacy PLC programming is not supported here (e.g. Ladder Diagram). SysML is based on four main concepts:

- **Requirements**

The requirement diagram collects requirements hierarchies

and the derivation, satisfaction, verification and refinement relationship. The requirement diagram is the bridge between typical requirements management tools and the system models.

- **Structure**

The structure of the system is developed using block definition diagrams and internal block diagrams. As seen for other standards, the block is the basic unit used to represent a system element, like a piece of hardware or software. The former diagram, the *block definition diagram*, has the purpose to describe the system hierarchy while the latter, the *internal block diagram* describes the internal structure of the system describing its interconnections (e.g. ports and connectors). The organization of the model is depicted in the package diagram.

- **Behaviour**

The behaviour is mainly represented using four diagrams, all inherited from UML 2:

- **Use case diagram:** provides a high-level description of functionality achieved by integrating multiple systems or system parts.

- **Activity diagram:** represents how data flows within the system, providing an idea of the control flow between activities.
- **Sequence diagram:** represents the interaction between various parts of the system describing how they collaborate.
- **State machine diagram:** handles the control flow of the system, responding to external events by changing the internal state of the system.

- **Parametrics**

The parametric diagram defines constraints on acceptable system metrics such as performance, reliability and other properties to support engineering analysis.

SysML extends/modifies UML to fill its gaps in representing some of those concepts as depicted in Figure 2.3. The *Block definition diagram* is redefined by replacing the concept of classes with blocks, and introducing flow ports which are defined in SysML as what can go through a block whereas it is data, matter or energy. The *Internal block diagram* is redefined by adding support to blocks and flow ports, while the *Activity diagram* has been modified to allow disabling of actions that are already executing

(which was not possible in UML, where control can only enable actions to start).

The *Parametric diagram* is intended to support system analysis (performance, reliability, etc.) by defining constraint blocks. A constraint block expresses a mathematical equation and its parameters, some of which may correspond to system block properties.

Finally *Requirement diagram* defines a visual and graphical representation of textual requirements, specialised associations between themselves or with other elements of the model, and how they can be managed in a structured and hierarchical environment. [11].

Thanks to all these types of diagrams, it supports the analysis, specification, design, verification and validation of complex systems such as hardware or software. All these diagrams provide a representation of the system from different points of view and due to the hierarchical structure of SysML it's possible to extend it through the creation of profiles and can be integrated into existing tool environments. Secchi et al. [46] have introduced the use of UML in automation with an example methodology of PLC code design by refining UML specifications.

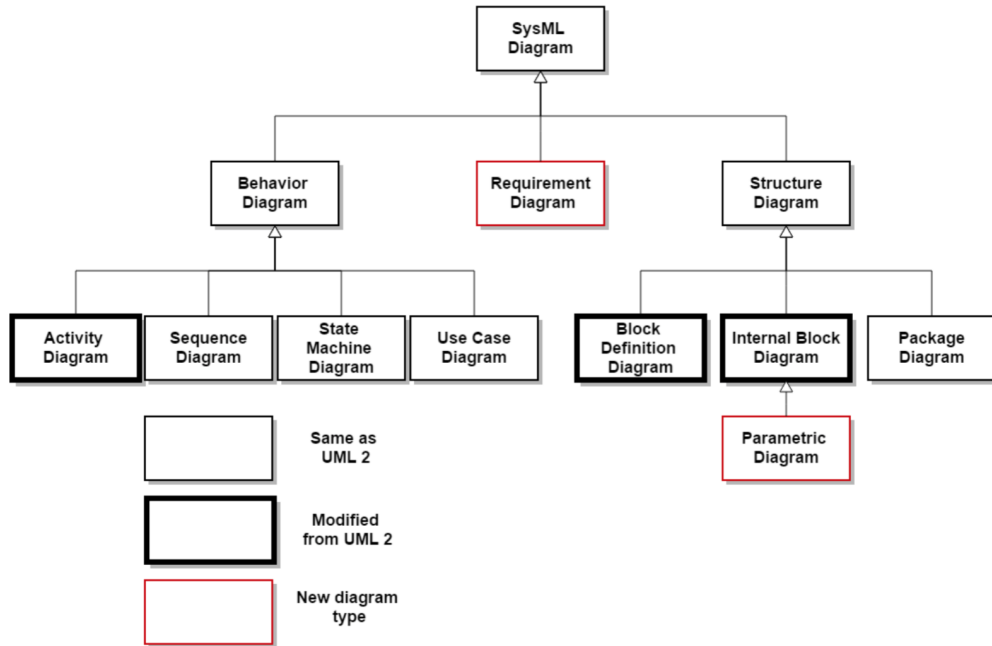


Figure 2.3: SysML diagram taxonomy as presented on the official website www.omg.sysml.org

Petri nets

Formal models of software systems can create dependable software with properties that can be enforced and guaranteed by design. In automation systems in fact, the dynamics of the plant is important and must be taken into account when control software is designed and developed. Additionally, the PLC code generation from a formal model is simple and has been investigated by many researchers [12], [13], [35]. Petri nets in this context are the most used tool for developing formal models. For example, Thieme and Hanish [49] developed an appropriate modeling for-

malism and suitable methods for automatic generation of modular control code for Programmable Logic Controllers (PLCs). This methodology is based on a formal model of the controlled object (plant). First the plant model describing the physical possible behavior is designed using a modelling formalism that the authors defined as *Net Condition/Event systems* [15] which is an extension of Petri Nets [41]. Then the same formalism is used to define a modular structured model of the controller. These two modules are then interconnected via their interfaces to establish what the closed loop behavior will be. After verifying that the closed loop model works, the controller code is automatically generated by translating modules in function blocks. These function blocks and their interactions reflect the structure of the modules and their interconnections with the controller model. Another similar approach to automatically generate PLC code from Petri nets has been developed by Music et al. [39]. An interesting comparison between PLC ladder logic and petri nets is presented in [53]. All these approaches of formal models which are theoretically promising, are not yet common in industrial practice among automation and control engineers because their application is associated with high computational complexity.

Multi-Agent Architecture

Multi-Agent architectures offers a new alternative to design decision-making systems. Instead of relying on a single controller which manages all the process, multiple intelligent entities collaborate to coordinate their functions within the overall system. Research in this field has been subject of several surveys [32] and is gaining practical usage in industrial applications [38].

Agent-Based Control (ABC) is often applied in the context of Reconfigurable Manufacturing Systems (RMS) research since it properly suits the needs of RMSs. An agent is a software component that has a general interface to external systems, and it is characterized by:

- **Autonomy:** It can perform tasks independently without intervention of humans or others.
- **Communication:** It can interact and communicate with other similar entities to ask for help achieving its own goal, or supporting others in achieving their goal. In this sense an agent is social [42].
- **Pro-activity:** It can take initiative to undertake an action without being explicitly triggered by a command from the user.

The idea of having multiple intelligent entities, has a good reflection in modern software development techniques like OOP (Object-oriented programming) where multiple objects interact each other to fulfill a common objective. All these objects have its own internal logic which given the provided input, and the internal status, can provide a decision as output. These applications are useful also for cases where a remote controller does not have enough bandwidth to transmit every single control command. In these cases, the remote controller is just a supervisor which gives objectives to every single entity, which can take decisions on its own to fulfill it. Even if this architecture looks promising, existing agent platforms do not always satisfy the real time requirements of practical automation applications as stated by Theiss et al [48] leading to significant overhead in respect of design effort and runtime resources.

The proposed approach in this thesis tries to take the benefits of all the aforementioned methodologies, and apply such benefits into a new method specifically meant for custom machines, which software is programmed on microcontrollers and integrated circuits. All the process described, is meant to be used across domains, from the design of the machine, through the assembly, to the software development and deploy, providing a single point

where all the various engineers can work on, sharing the design of the machine and working on it along the whole process.

2.2 Remote maintenance

The main idea of remote maintenance is that, by leveraging information, wireless and internet technologies for communication and transport, technicians can remotely log into machineries to analyze their behavior and reading operation configurations, other than performing set-ups and configurations [18]. This allows to reduce the manufacturer's manpower retained on the customer's site, allowing remote analysis of issues, and progressive improvements on the preventive maintenance algorithms thanks to the machine-performance monitoring.

2.2.1 E-Technologies for maintenance improvements

The combination of modern processing techniques and communication tools offers the technical support for remotely accessing information and process. Such technologies allow to transfer system and environment knowledge to maintenance specialists remotely located in order to operate together through remote exchange [26].

First and major enabling technology for this sector is the Web. With all its open standards and support for different platforms, it enables easy communication, messaging, and networking between devices and operators [55]. Database technologies allow to store data, which can be collected by legacy systems, for offline analysis or monitoring.

Next, wireless technology allows in some cases to reduce the wiring, reducing costs, and allowing more flexibility and availability of information on the factory floor. This allows data reading from machines with mobile terminals, and data sharing and machines control from anywhere in the world. The Internet in this case provides a great transport layer for communication between the factory and the remote controller [40].

These e-technologies increase the possibilities to utilize data from several sources and of various kind, to process large amount of data to support decision making and production prediction and to share information between machines for implementing collaborative activities.

These technologies are also enabling factors for cooperative/-collaborative maintenance [27]. An information infrastructure can in fact connect geographically dispersed systems or actors like suppliers, clients, support engineers, leveraging the Internet net-

work. E-maintenance introduces a great level of transparency in the whole organization, drastically reducing interfaces between personnel, departments, or IT systems. It allows synchronization between maintenance and production, buyers and sales, minimizing downtime costs. All this transparency, facilitates the bidirectional flow of data from the factory floor to decision levels, automating the retrieval of information that decision makers require to schedule maintenance activities in an optimized fashion [51].

All the data collected thanks to these technologies, can be used for developing predictive maintenance [37] algorithms and progressively improve on them. E-maintenance provides companies with intelligence tools to monitor their machineries anticipating potential breakdowns. Product's performance can be monitored over time, allowing the company to focus on degradation monitoring instead of discovering and recovering faults [25].

2.2.2 E-maintenance

Remote maintenance in literature has been explored mostly under the concept of e-maintenance [33]. E-maintenance is a wider field where all the E-technologies explained in the previous section are used for supporting maintenance operations. Various standards

have been developed for supporting e-maintenance platforms/architectures. The main are:

- IEC 62264 (enterprise control system integration) [20]
- ISO 15745 (industrial automation application integration framework) [24]
- MIMOSA (Machinery Information Management Open System Alliance) [36]
- ISO 13374 (condition monitoring and diagnostics of machines) [23]

2.2.3 Web technologies as enabling factors for remote maintenance

The ease of access to the Internet, through standard communication protocols that provide the transfer of data throughout the world, represents the basics of the ability to perform maintenance and monitoring on a remote machine. One of the first articles introducing the concept of remote maintenance for globally integrated manufacturing facilities is [29] which proposes functional requirements for remote maintenance systems:

- **Multi-sensor Integrated Monitoring and Control Systems** Monitoring requires multiple sensory devices to acquire data from the plant
- **Communications and Integration** A multimedia information network is a basic requirement for transferring and sharing information among geographically dispersed participants.
- **Data Abstraction** Data transmission is the key concept, and performance of transmission matters. Data has to be compressed and possibly aggregated to provide useful real time insights.
- **Tele-Maintenance and Collaborative diagnostics** Multimedia-based tools are the mean to support remote users for maintenance assistance. Interactive and collaborative tools enables technical personnel to operate diagnostics from a remote location.

In [31] standardized and distributed measurement and control frameworks are illustrated, while in [30] an approach to web enabled e-maintenance systems is presented. A literature review of web and agent technologies in condition monitoring the review

shows that Web and agent technologies are being used for monitoring and maintenance in manufacturing, power, and chemical industries. It is used to integrate geographically distributed systems, processes and heterogeneous data for asset management. In [54] an Internet based monitoring system to monitor the performances of distributed power stations is presented. The idea to use mobile devices as support tool on the factory floor in the field of maintenance was already explored in [5] with an interesting example of a mobile maintenance support system based on web and mobile device technologies, i.e., personal digital assistant. Mobile devices are also considered in [9] within an application for engineering asset and maintenance management.

In [1] is presented an approach to integrate Internet technologies in maintenance processes in a project called PROTEUS [2], with the objective to improve maintenance processes efficiency by connecting the operator to the expert via the Internet. Another example of using the Web as a maintenance portal is described in [57] where two implementation examples for a web maintenance portal are presented.

In [14] the design choices of a flexible E-maintenance platform are discussed. The platform is then built based on a combination of web services and statistical analysis to obtain an agile integra-

tion of maintenance activities with the production process.

Finally, [56] presents a good example implementation of a remote maintenance system based on XML documents, which publishes data on the web and alarms upon specific values, by sending messages to engineers mobile devices. On the same stream, a condition monitoring system is presented in [7] describing how a monitoring system can be used to constantly check the status of components that work in harsh environments and that cannot be visually inspected. The system presented augmentates the condition data with risk information to improve the maintenance activity.

Despite the efforts spent in this wide field of the remote maintenance, the analysed literature didn't show any specific approach for semi-automated machines. The usage of a smart device as the primary tool for remote maintenance of industrial machinery and plants is one of the main innovations that this thesis brings: these terminals, in fact, provide in a single device the latest technologies in human-user communication and localization at a very low price. Currently, monitoring and remote maintenance systems are essentially based on dedicated hardware; this choice, although necessary in the past (due to the lack of integrated devices as MID), at this time is increasingly inefficient, largely because of the rapid

obsolescence of dedicated devices.

Chapter 3

Integrated methodology

In this chapter, the two methodologies, contributions of this thesis are presented. In the first section, the focus is on how to support reconfigurability and multiple variants in software development for industrial machines. The concept of module will be the key to support a fast reconfiguration and regeneration of the entire software artifacts. In the second section, later in the chapter, the integrated approach for remote maintenance using modern smart devices is described.

3.1 Automation software development method for semi-automated machines

Considering that a large part of manufacturing systems are realized assembling various mechatronic objects on the shelf it is quite straightforward to think about having a similar concept of pre-assembled objects for automation software too. This is what this development method for software development aims at.

A manufacturing system in fact (as done in most of the companies) can be considered as an agglomeration of already built mechatronic blocks with infrastructural and interconnection components that allow physical and logical communication between them. The introduced approach aims at replicating this modularity in the automation software with the purpose of having a perfect corresponding between hardware modules and software modules. The rationale behind this choice is that considering that mechatronic modules are not changing, it does not make sense to write the automation software from scratch each time. It is a wiser investment instead to rethink the way the software is developed once, and take advantage from modularity from then on. If every mechatronic object is packed with a software modules that allows an high level interaction with the hardware by implementing low

level hardware functions, is then easy to build the automation software for the entire system just by assembling those modules.

The proposed methodology (as depicted in Figure 3.1) describes how to structure a software module, which process to follow for allowing an easy assembly of such modules, and how to automate some operations that are linked to software development and that deconcentrate the developer from his main job: writing good software.

3.1.1 The concept of module

The main idea of the software module is to provide encapsulation and information hiding, concepts typical of high level software languages by exposing a defined interface for invocation and communication with it, without exposing the internal implementation of the module. The development of black box modules, facilitates testing and verification because the software engineer can rely on already tested and validated modules. This allows the developer to concentrate on a smaller but important part in software development: the machine working cycle. This core part of the software is obviously different from machine to machine and describes how each module interacts with the other ones, and it is the most valuable and critical part of the machine automation

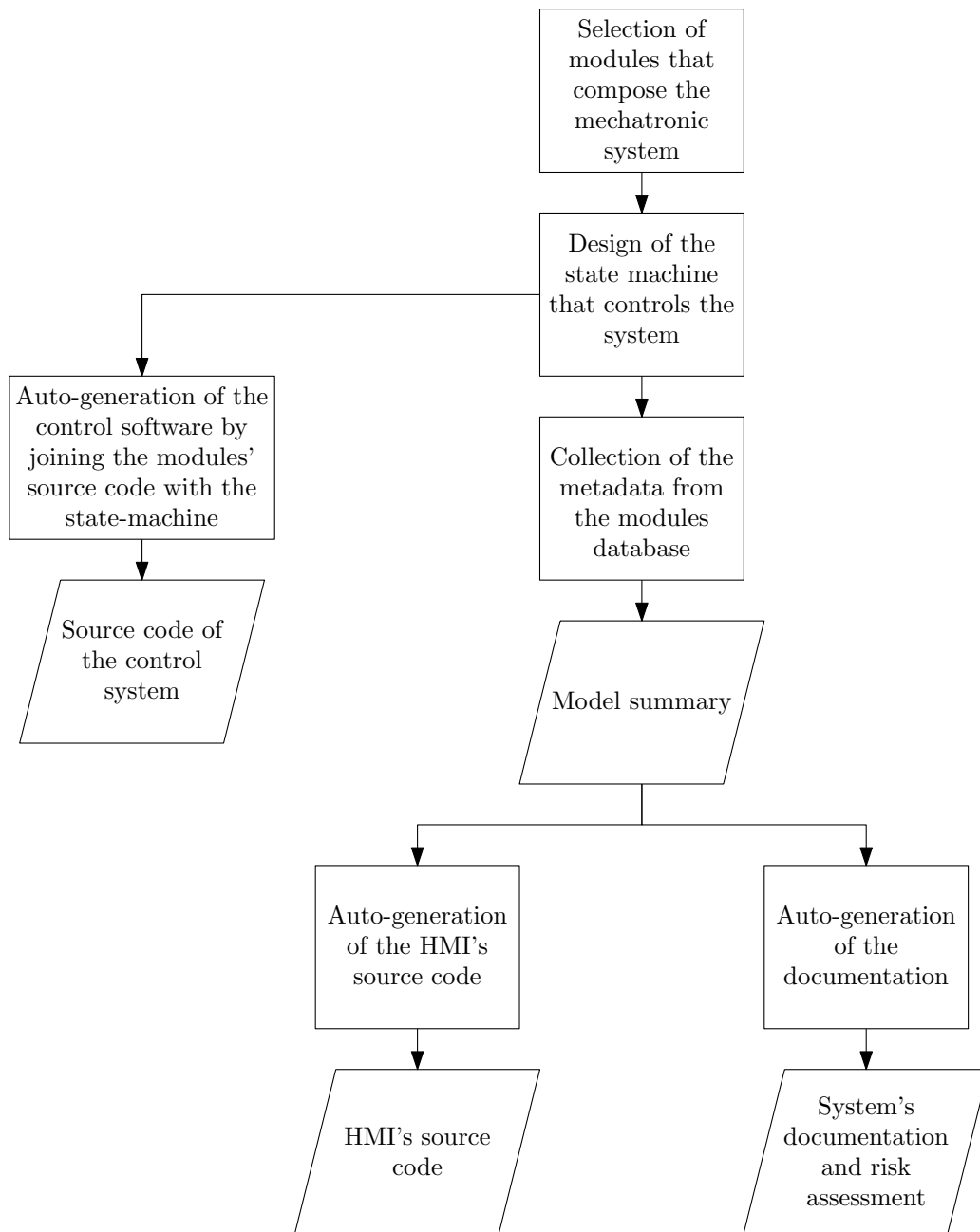


Figure 3.1: Helicopter view of the proposed methodology

software. Still with the purpose of simplifying the work of the developer and making it less error prone, other parts of the de-

velopment can be automated too. Depending on what modules are loaded on the machine, the user documentation of the system will have certain sections, the testing checklist will have specific constraints to be met, HMI (Human Machine Interface) will have dedicated menus to allow configuration of those components. To automate the generation of these artifacts, all this information has to be associated to the module itself. The identified minimal set of information chunks needed for the process is:

- **Versioning information** Each module release is marked with a specific version number. This is useful to keep track of features and issues associated with each module. When trying to understand if a specific machine is affected by a bug just discovered, is sufficient to check if the version of the module that the machine runs is the one affected by the defect or not.
- **Description** A brief description that states what the module does, and which hardware component it maps to
- **Documentation** The pieces of the user manual that have to be included in the documentation if this module is used on a specific machine. These documentation parts are in multiple languages to allow automatic generation of multi-language

user manuals.

- **Interfaces** The definition of how it is possible to interact with the module, what are the actions that the module can be asked to accomplish, and how to read data from it. These interfaces are the interconnection point between the machine cycle and the black box module internals. Interfaces of the module can be either physical or logical. Physical interfaces are the physical pins that the module exposes for wiring. Logical interfaces instead are software interfaces that are used to make the module interact with the rest of the automation software. The connection between logical interfaces and physical interfaces is depending on the wiring made by the electricians.
- **Configurable properties** Configuration values that can be set when instantiating the module to parameterize its work. For example there can be values like maximum torque, maximum load, etc. that can depend on the specific machine structure, so the software should be parametric to allow these values to be taken into account during execution. These properties, are set once the module is instantiated, and are not configurable at runtime. It's the vendor parameterization

of the machine.

- **Risks** Some mechatronic modules can be dangerous for humans during their operation. The risks associated with a specific modules are listed here, with a description of each. This risk information can be used to automatically generate a risk assessment document.
- **Runtime configurable parameters** Each module can have associated configuration values that should be allowed to change during normal machine operation through the system HMI. This can be required for example to set the force that a gripper has to use to pick up components, or the distance that the machine has to keep from the object it's painting.

All these modules are stored in a database that can be accessed each time a module is needed for development. It's a library of pre-built software modules that can be used to assembly the machine software. Such a database guarantees:

- The usage of the latest validated and verified version of the module. This allows the user to use a component that has already been tested, instead of relying on copy-paste techniques widely used in the industry. Having components fully tested gives the overall machine automation software a high

test coverage score by default, leaving to the developer the responsibility of only the integration tests.

- An high traceability of software defects. When an issue is identified in a specific module version, it is straightforward to find all the systems that uses that specified buggy version and ship an update for fixing it. Updating all the fleet of machines that have that defect is easy with modules, because of the concept of incapsulation. The software that interacts with the module sees it as a black box, and relies just on its interfaces. If the interfaces are not changed, the internal software of the module can be changed anyhow and replaced in a plug and play fashion. This is way easier compared to what would have meant to correct the defect in many machines where the code is not so well structured.
- An increase of hiererearchy levels in software development. The engineer that develops the software module is different from the one that implements the machine working cycle. This allows developers to focus on smaller parts of software, increasing the simplicity of the software and the reliability of the code.

This database is developed and maintained by the module soft-

ware engineers who are responsible that the mechatronic module is correctly managed by its software counterpart. They will be responsible of maintaining and improving the software, and releasing periodical updates that can be simply integrated by maintaining the interfaces unchanged.

The language chosen for the module definition is XML. The schema that the definition has to adhere to is:

```
1 <xs:schema attributeFormDefault="unqualified" elementFormDefault="
  qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
2   <xs:simpleType name="io_type">
3     <xs:restriction base="xs:string">
4       <xs:enumeration value="DINPUT" />
5       <xs:enumeration value="DOUTPUT" />
6       <xs:enumeration value="AINPUT" />
7       <xs:enumeration value="AOUTPUT" />
8     </xs:restriction>
9   </xs:simpleType>
10  <xs:element name="module">
11    <xs:complexType>
12      <xs:sequence>
13        <xs:element type="xs:string" name="parent_node"/>
14        <xs:element type="xs:float" name="version"/>
15        <xs:element type="xs:string" name="description"/>
16        <xs:element name="properties">
17          <xs:complexType>
18            <xs:sequence>
19              <xs:element name="property" maxOccurs="unbounded"
minOccurs="0">
20                <xs:complexType mixed="true">
21                  <xs:sequence>
22                    <xs:element type="xs:string" name="value"
maxOccurs="unbounded" minOccurs="0"/>
23                  </xs:sequence>
24                  <xs:attribute type="xs:string" name="name" use="
optional"/>
25                  <xs:attribute type="xs:string" name="type" use="
optional"/>
26                </xs:complexType>
27              </xs:element>
28            </xs:sequence>
29          </xs:complexType>
30        </xs:element>
```

```

31     <xs:element name="warnings">
32       <xs:complexType>
33         <xs:sequence>
34           <xs:element name="warning" maxOccurs="unbounded"
minOccurs="0">
35             <xs:complexType>
36               <xs:simpleContent>
37                 <xs:extension base="xs:string">
38                   <xs:attribute type="xs:string" name="
linked_property" use="optional"/>
39                   <xs:attribute type="xs:string" name="
property_value" use="optional"/>
40                   <xs:attribute type="xs:string" name="symbol"
use="optional"/>
41                   <xs:attribute type="xs:string" name="id" use="
optional"/>
42                 </xs:extension>
43               </xs:simpleContent>
44             </xs:complexType>
45           </xs:element>
46         </xs:sequence>
47       </xs:complexType>
48     </xs:element>
49     <xs:element name="risks">
50       <xs:complexType>
51         <xs:sequence>
52           <xs:element name="risk" maxOccurs="unbounded"
minOccurs="0">
53             <xs:complexType>
54               <xs:simpleContent>
55                 <xs:extension base="xs:string">
56                   <xs:attribute type="xs:string" name="
linked_property" use="optional"/>
57                   <xs:attribute type="xs:string" name="
property_value" use="optional"/>
58                   <xs:attribute type="xs:string" name="id" use="
optional"/>
59                 </xs:extension>
60               </xs:simpleContent>
61             </xs:complexType>
62           </xs:element>
63         </xs:sequence>
64       </xs:complexType>
65     </xs:element>
66     <xs:element name="checks">
67       <xs:complexType>
68         <xs:sequence>
69           <xs:element name="check" maxOccurs="unbounded"
minOccurs="0">
70             <xs:complexType>
71               <xs:sequence>
72                 <xs:element type="xs:string" name="item"

```



```

maxOccurs="unbounded" minOccurs="0"/>
73     </xs:sequence>
74     <xs:attribute type="xs:string" name="
linked_property" use="optional"/>
75     <xs:attribute type="xs:string" name="
property_value" use="optional"/>
76     </xs:complexType>
77     </xs:element>
78     </xs:sequence>
79     </xs:complexType>
80 </xs:element>
81 <xs:element name="io_symbols">
82     <xs:complexType>
83     <xs:sequence>
84     <xs:element name="io_symbol" maxOccurs="unbounded"
minOccurs="0">
85         <xs:complexType>
86         <xs:simpleContent>
87         <xs:extension base="xs:string">
88             <xs:attribute type="io_type" name="type"/>
89             <xs:attribute type="xs:string" name="help"/>
90         </xs:extension>
91         </xs:simpleContent>
92     </xs:complexType>
93     </xs:element>
94     </xs:sequence>
95     </xs:complexType>
96 </xs:element>
97 <xs:element name="app_symbols">
98     <xs:complexType>
99     <xs:sequence>
100     <xs:element name="app_symbol" maxOccurs="unbounded"
minOccurs="0">
101         <xs:complexType>
102         <xs:simpleContent>
103         <xs:extension base="xs:string">
104             <xs:attribute type="xs:string" name="type"/>
105             <xs:attribute type="io_type" name="connection"
/>
106             <xs:attribute type="xs:string" name="help"/>
107         </xs:extension>
108         </xs:simpleContent>
109     </xs:complexType>
110     </xs:element>
111     </xs:sequence>
112     </xs:complexType>
113 </xs:element>
114 <xs:element name="parameters">
115     <xs:complexType>
116     <xs:sequence>
117     <xs:element name="parameter" maxOccurs="unbounded"
minOccurs="0">

```

```

118         <xs:complexType mixed="true">
119             <xs:sequence>
120                 <xs:element type="xs:short" name="value"
maxOccurs="unbounded" minOccurs="0"/>
121             </xs:sequence>
122             <xs:attribute type="xs:string" name="type"/>
123             <xs:attribute type="xs:string" name="
linked_property"/>
124             <xs:attribute type="xs:string" name="
property_value"/>
125             <xs:attribute type="xs:string" name="var_name"/>
126             <xs:attribute type="xs:string" name="position_code
"/>
127             <xs:attribute type="xs:short" name="value_max"/>
128             <xs:attribute type="xs:short" name="value_min"/>
129         </xs:complexType>
130     </xs:element>
131 </xs:sequence>
132 </xs:complexType>
133 </xs:element>
134 <xs:element name="documentation">
135     <xs:complexType>
136         <xs:simpleContent>
137             <xs:extension base="xs:string">
138                 <xs:attribute type="xs:string" name="id"/>
139             </xs:extension>
140         </xs:simpleContent>
141     </xs:complexType>
142 </xs:element>
143 </xs:sequence>
144 <xs:attribute type="xs:string" name="name"/>
145 </xs:complexType>
146 </xs:element>
147 </xs:schema>

```

Listing 3.1: Schema of a Module definition

The selected environment is Matlab Simulink due to its graphical workspace which intrinsically implements the concept of Block and due to its ability to automatically generate code which is supported by several manufacturing systems. Each module is developed into a Simulink block with the same interfaces defined in the XML document. These modules are then stored in a Simulink

library. Once the library is built, the developer can select blocks from the library each time the same mechatronic module is present on the machine to compose the machine software, and concentrate on the development of the machine working cycle.

3.1.2 The process

Once the various software modules have been developed and stored in the software database, software engineers specialized in machine working cycle development can use them for their programs. The process designed for the development is the following:

Selection of software modules

During the design of the machine, electrical and mechanical engineers work together to identify which hardware modules are needed for building the machine for the client. Once the selection is completed, they start to put on stage on a computer application those modules. This allows them to track easily which modules and modules versions are required. All the used modules are stored on a file (from now on *Project file*) that will be passed on to the electricians that will be responsible of wiring the machine. This file in fact, will be carried on during the whole development process and will serve as a tracking archive for the

whole process.

Wiring of the machine

Once the design has been completed, the mechatronic assembly takes place. Electricians connects various modules physical interfaces with the central processing unit. When wiring the modules, the Project file is updated to reflect the connections made on the physical machine. As said, every module has both physical and logical interfaces. As an example, a gripper module can have two software interfaces (motor forward, motor backwards) that can be connected to two physical interfaces depending on where the gripper motor is soldered. This association between logical interfaces and physical pins is made by electricians that are wiring the machine. This is not designed upfront, because most of the time the wiring is made depending on how the various components are positioned and can be changed during setup depending on the space available. Thanks to this connection of logical/physical instances, when a software variable is set, an high voltage level is sent to that specific pin.

Design of the working cycle automation software

The design of the automation software for defining the machine working cycle is the place where the developer effort is best spent. The working cycle in fact is the real only part that changes in each developed machine, and this is where the focus of the developer should be. The software for controlling modules is already available within the modules, so the only part remaining to develop is the behavior of the machine. The proposed approach for designing a working cycle is to develop it as a state machine. Every state represents a machine working condition and in that state machine actuators can be controlled by interaction with the modules. Transition conditions can depend on values read from the modules and internal states. The developer can start working on the file provided by the electricians after they wired the machine. This file already contains all the modules with the correct link on the hardware modules interfaces. The developer has to develop the state machine reading values and sending commands to the modules to invoke actions on them. The state machine is designed using Matlab Stateflow, which allows to represent state machines and wire inputs and outputs to external simulink blocks. The main features of Stateflow are listed here:

- Modeling environment, graphical components, and simula-

tion engine for modeling and simulating complex logic.

- Deterministic execution semantics with hierarchy, parallelism, temporal operators, and events.
- State diagrams, state transition tables, and state transition matrices representing finite state machines.
- Flow charts, MATLAB functions, and truth tables for representing algorithms.
- State diagram animation, state activity logging, data logging, and integrated debugging for analysing the design and detecting runtime errors.
- Static and run-time checks for transition conflicts, cyclic problems, state inconsistencies, data-range violations, and overflow conditions.

The state machine, can then be tested on its own in a separate test harness which can simulate inputs and outputs to the state machine itself, and verify that the outputs are as expected. This test harness has to be set up for every state machine as the expected behavior is different every time. This can help to verify that the machine responds correctly, verify deadlock conditions,

and can be used to generate Condition Coverage/Decision coverage reports to see if the test environment validated all the possible cases and branches of the state machine or not. Once the state machine is validated, the code can be generated using MATLAB Coder. The generated code wasn't verified as part of this work, but it never demonstrated inconsistencies in behavior compared to the Stateflow implementation. A potential improvement of this approach can be to derive tests from the state machine according to coverage criteria, and then executed on the generated code to validate the compliance of the code with the model.

Automatic generation of automation software

Once the development of the machine working cycle is complete, the code generation can be started. This is achieved using code generation tools that allow to generate source code from the state machine, and join it with the modules source code. The resulting code is then compiled together with a base firmware that is shared among all the machines and realizes the abstraction layer that allows linking between hardware interfaces defined in the modules and the physical pins on the electronics.

Collection of Metadata from the modules database

The modules database is read looking for definitions of the modules loaded in the current project, and the corresponding module metadata are loaded. This metadata contains all the information needed for automatically building the documentation, risk assessment documents, and HMI source code. Since these features are implemented by external tools, we need to export a document that describes the system designed in Simulink. This descriptor is called *model summary* and it is generated by a script that is able to read from the designed system model and generates this file in a text format (XML has been selected for this purpose). The schema of this XML model summary is represented below:

```

1 <xs:schema attributeFormDefault="unqualified" elementFormDefault="
2   qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="modules">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element name="module">
7           <xs:complexType>
8             <xs:sequence>
9               <xs:element name="property">
10                <xs:complexType>
11                  <xs:sequence>
12                    <xs:element type="xs:string" name="value"/>
13                  </xs:sequence>
14                  <xs:attribute type="xs:string" name="name"/>
15                </xs:complexType>
16              </xs:element>
17            </xs:sequence>
18            <xs:attribute type="xs:string" name="name"/>
19          </xs:complexType>
20        </xs:element>
21      </xs:sequence>
    </xs:complexType>

```



```
22 | </xs:element>  
23 | </xs:schema>
```

Listing 3.2: Schema of the model summary

This XML contains all the modules which are currently loaded within the model along with the values of their properties that has been set by the engineer. These properties values are important because depending on the properties there may be different documentation to show, different checks to be performed in the security checklist or more or less risks to be reported. This file is the starting point for the execution of the next steps.

Automatic generation of the HMI software

Depending on the modules that are available on the machine, 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. Through the HMI, the operator of the machine will be able to set the configuration parameters that were previously defined in the modules metadata. The parameters setting and reading interfaces are all built automatically starting from the definition in the modules metadata. Figure 3.2 represents the composition of the machine automation and HMI software.

At the end of the process, the packed automation software is

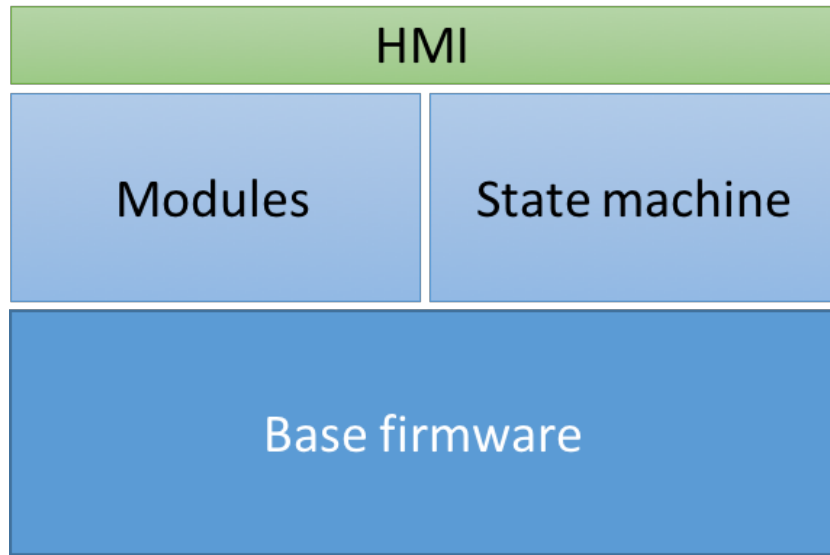


Figure 3.2: The composition of machine automation software

installed on the machine, and documentation is ready for manual refinement and additions, before delivering it to the customer. Figure 3.3 describes the flow of artifacts from the model to the final software and documentation.

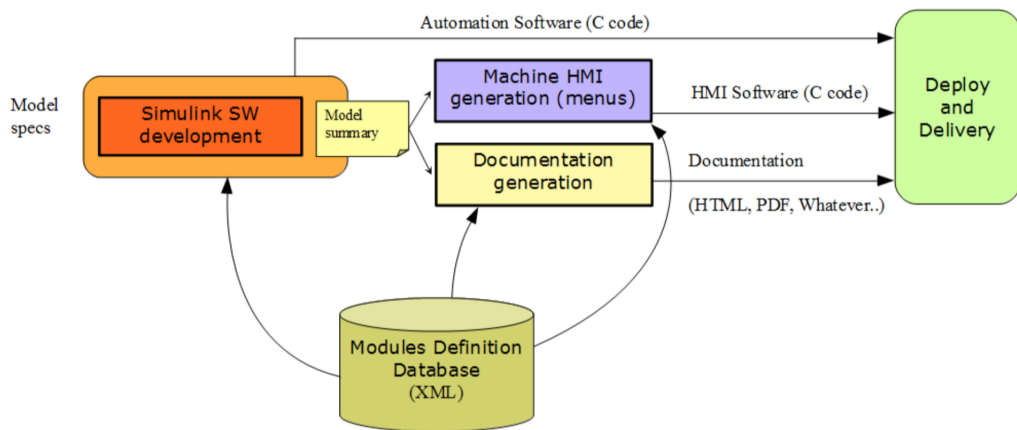


Figure 3.3: The process for generation of the machine software

Automatic generation of 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 composing 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 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. The documentation generator tool developed reads the model summary and generates all the documentation source code in the *Doxygen* format. This format can be parsed by the Doxygen tool to generate PDFs or other documentation formats. The data for generating these documents is taken again from the modules definition, and can support multilanguage if the modules are defined in more than one language.

3.2 Development of a remote maintenance system for semi-automated machines

The proposed solution aims at providing a way to debug issues on remotely located semi-automated machines, based on the acquisition of both data and video streams of the running machine. In most of the cases, especially when talking about fully automated machines, the machine log is enough to be able to monitor and debug issues or wrong behaviors that the machine manifests. In the case of semi-automated machines instead, there is an additional noise source, which is the human interaction. This interaction can sometimes be seen in the machine logs, but sometimes there aren't enough sensors on the machine to be able to trace or identify the human intervention in the mis-behavior manifested by the machine. This can be the case especially when the reduction of costs policy requires a reduced amount of sensors to be available on the manufacturing system, a reduced capacity processor, memory, I/Os. This may lead to have semi-automated machines where the user interaction can not be identified on the logs, requiring additional information to be captured in the debugging process. When an issue manifests, and the operator is able to reproduce

it, this system will allow to capture the user interaction together with the machine data stream, allowing engineers to understand when, and in which conditions, the machine manifests the wrong behavior. This is mostly the case where the human interaction is not purely digital (buttons or commands from the UI) but is also physical and analog. In this case the user is part of the machine movement, and interacts with items that the machine works with, altering the environment from the outside, with or without the machine knowing it.

The presented approach can be applied in cases where the machine manifests a wrong behavior in particular conditions, where the operator is able to reproduce the machine misbehavior. When this happens, the machine interaction can be recorded with this system while the data stream from the machine is acquired and sent along with the video to the service center of the vendor. A further development of the proposed system may be considering a setup for monitoring purposes, where the system continuously acquires a video stream of the machine usage so that a video/data log is always available also for issues that manifests only once but can't be reproduced because the triggering conditions are not yet identified. This further development can bring concerns about privacy because it entails a continuous video recording of opera-

tors on their work conditions, which have to be considered and evaluated, but are outside of the topic of the present work.

3.2.1 System architecture

The system architecture for this remote maintenance system has been designed to be modular. This will allow its application on a number of different machines as well as its retrofit on machines already in service. The proposed approach has one component that is needed to be developed ad-hoc for every machine, in order to make the data acquisition from the machine easy and abstract the specific acquisition logic behind a transmission protocol over Wi-Fi that this piece has to expose. This setup can be easier on certain machines (especially where a broadcast bus is available to probe) or more complex on some other machines, where the internal machine communication is not achieved through broadcast bus, but requires additional probing to be instrumented on the system.

The overall architecture is presented in Figure 3.4 and it is composed of the following components:

- **Machine** The machine that needs to be monitored. This machine has its own control electronics and interfaces. It can be a modern machine or a legacy one.

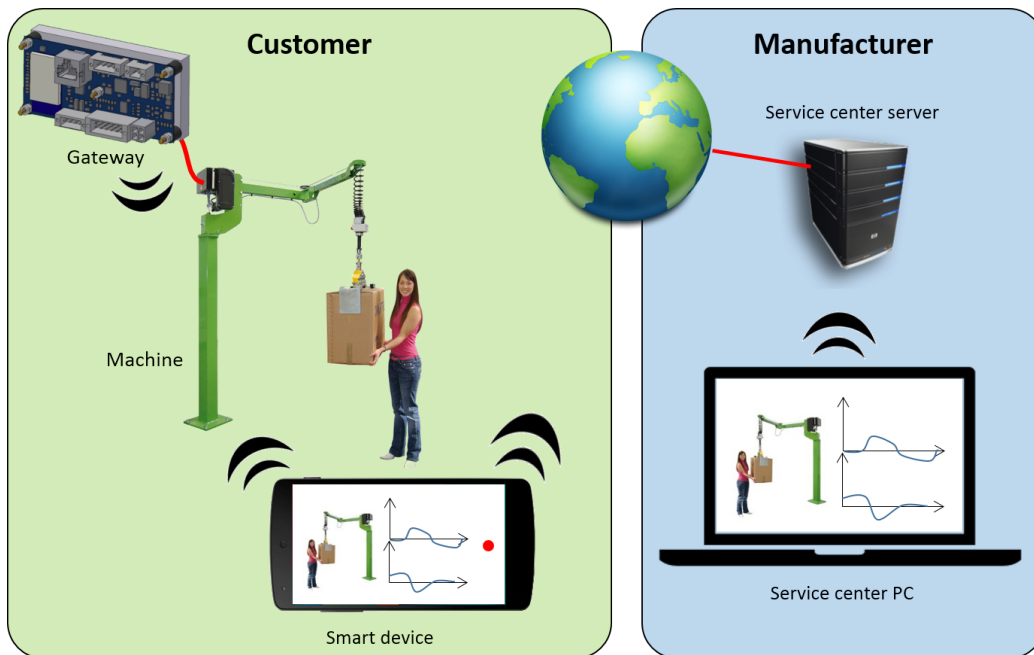


Figure 3.4: The architecture of the remote maintenance system

- **Gateway Hardware** The piece of hardware that allows streaming of the machines operating parameters over a wireless connection. It has the task of acquiring signals from the machine (in whatever format they are available) and transform those signals in digital ones that can be streamed over Wi-Fi. This component is the only part of the system that has to be machine specific. A smart device connects to it for gathering this data;
- **Smart device** The smart device can be any smartphone or tablet which has enough computational power to download data and record video simultaneously. Its main tasks are:

1. Connect and disconnect to the Gateway;
 2. Send *start* and *stop* commands to the Gateway to enable or disable the streaming of data;
 3. Acquire live data from the machine, through the wireless gateway;
 4. Record a video of the operator interacting with the machine, and store it together with all the acquired data;
 5. Package and route the synchronized video and operating data from the remote customer location to the machine manufacturers headquarters;
 6. Serve as a smart HMI to reconfigure the machine and change its behavior, once a solution has been devised;
- **Smart device app** The application that runs on the smart device for acquiring and sending data; This is installed on the smart device and it depends on the smart device operating system.
 - **Network infrastructure** A suitable network is required to send the acquired data to the manufacturer. In the most common case, the suggested method is the Internet;
 - **Service center server** The server at the machine manufacturer's headquarters that accepts all the data from remote

machines storing it in a structured way, accessible by diagnostic technicians. This server is connected to the Internet and it's the main endpoint for the connection of the Smart Device.

- **Service center PC** A diagnostic technician will get the recorded data on his/her PC in the service center. This PC displays the data together with a recording of the operator interacting with the machine, allowing the technician to analyze and try to debug the problem.
- **Service center PC app** The application that allows the diagnostic technician to review the acquired data in a synchronized way.

3.2.2 Remote assessment process

Figure 3.5 represents the process of recording data and video of the machine usage.

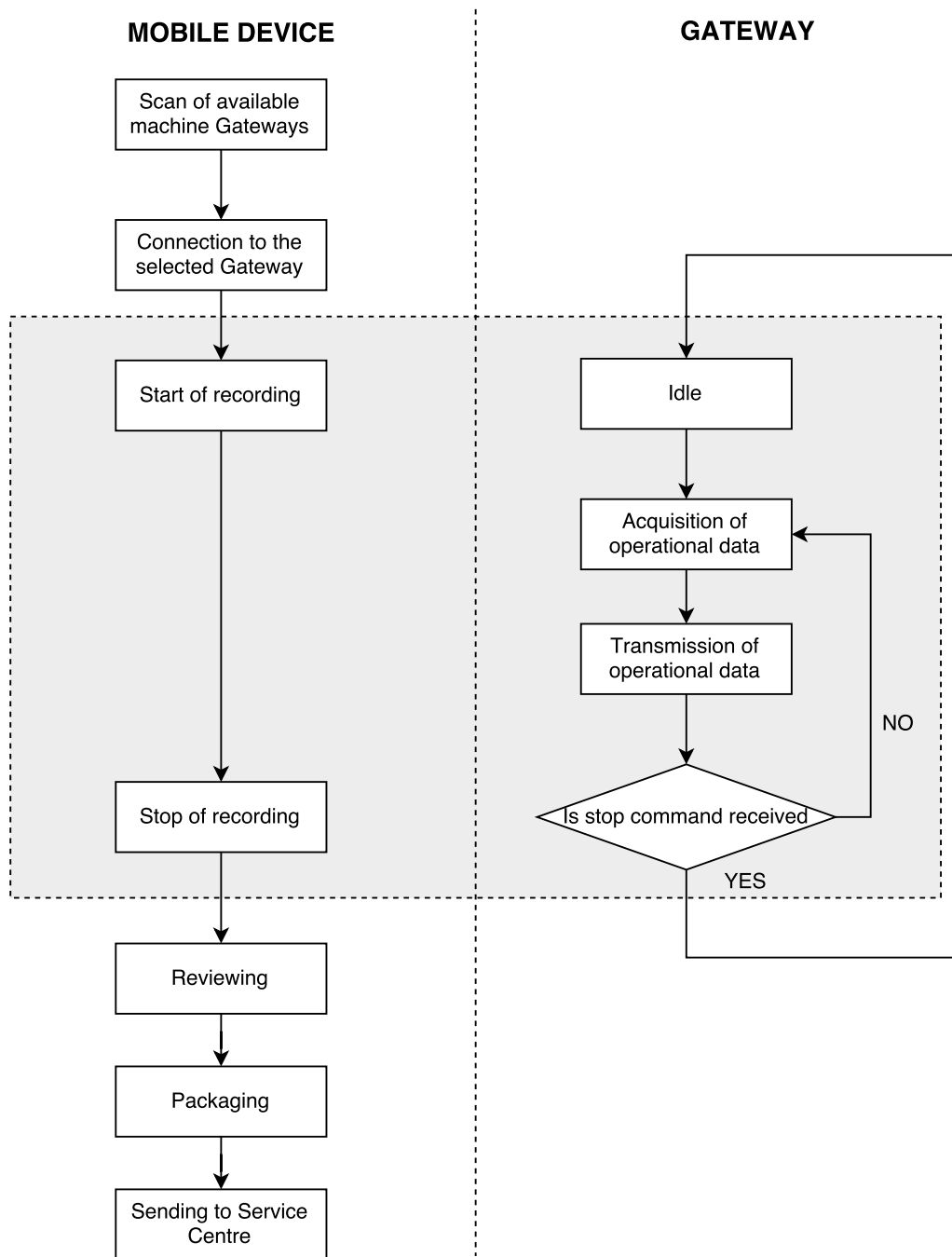


Figure 3.5: The data acquisition process

First, the user scans wireless networks for available gateways through the application on his smart device. Once he found the machine he wants to connect to, he can activate the connection. The application then interacts with the gateway to establish a Wi-Fi communication channel. Once the channel is setup, the user can start the recording of a video of an operator interacting with the machine. When he starts a video recording, the mobile application communicates with the gateway asking it to start acquiring data from the device.

When the gateway receives the *start* message, it starts acquiring data from the machine bus and sending to the mobile device until it receives a *stop* message. The device in the meanwhile starts to recording a video using the smart device video camera. All these information (both video and data) are stored on the smart device internal memory.

Once the user is satisfied with the recording, he stops the data acquisition and the *stop* message is sent to the gateway. After the acquisition is complete, the user can review the acquired data using the smart device, and if satisfied with that he can send it to the machine manufacturer service centre. To send the package, the application compresses it, disconnects from the gateway, connects to the corporate network to gain Internet access, and delivers the

package uploading it to the service centre server.

Once the package has been uploaded to the server, service center technicians can download it and analyze the data while seeing the recorded video. If the issue can be solved by a software change, the technicians corrects the bug and releases a firmware upgrade for the machine, and the upgrade process is started as depicted in Figure 3.6.

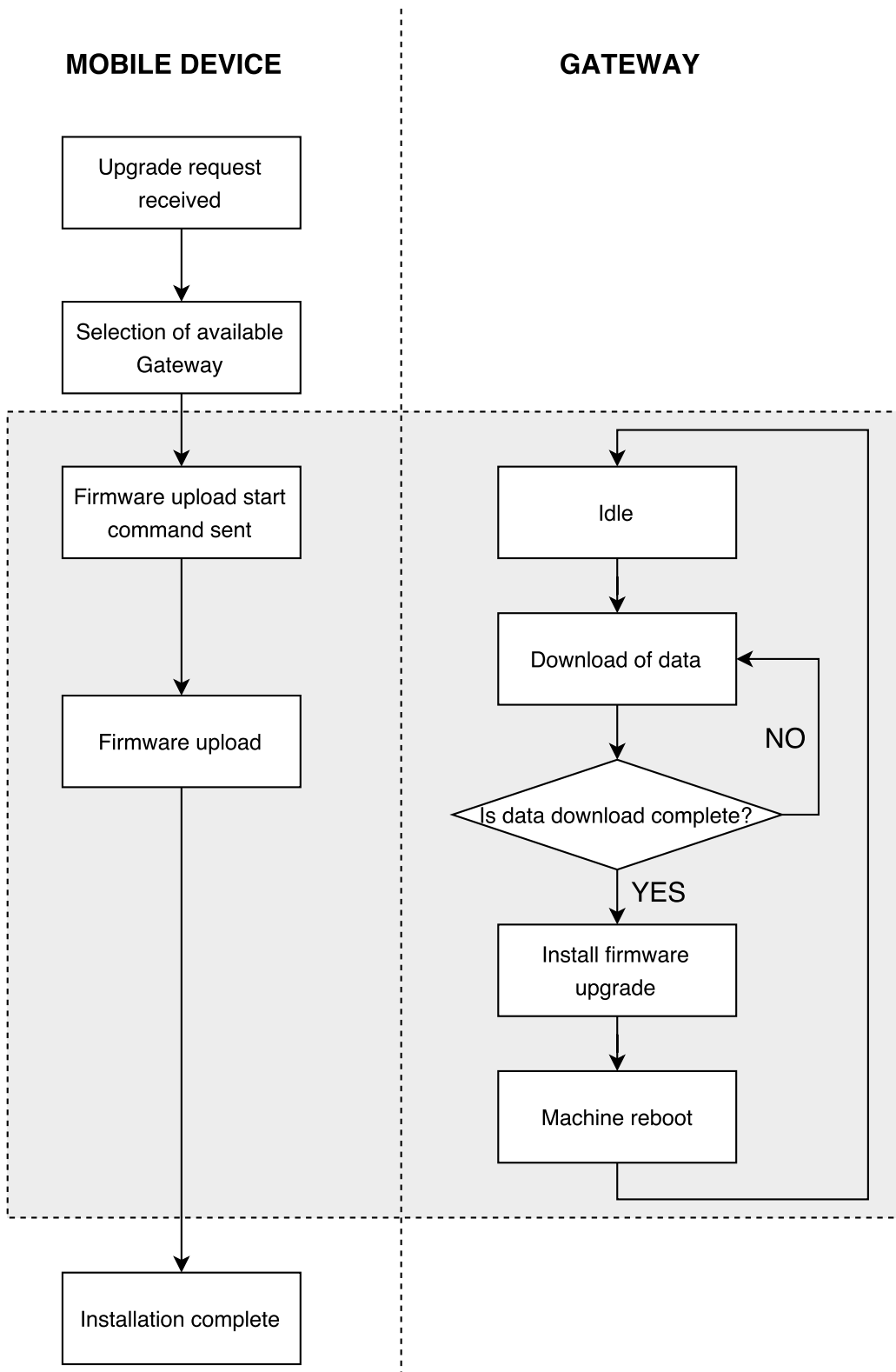


Figure 3.6: The firmware upgrade process

The mobile device receives a notification regarding an upgrade available for the machine. When the user opens the notification, he is asked to select a gateway connection among the discovered one. The user selects the machine that needs the software upgrade. Once the gateway is selected, and the connection established, the smart device starts uploading the firmware data towards the gateway. When the upload is complete, the gateway installs the upgrade by replacing the old firmware with the new firmware. The gateway then reboots the machine, and notifies the smart device about the completed upgrade.

3.2.3 Smart device app

The presented approach is based on a simple application Android-based. The reason for choosing Android as operating system compared to other alternatives has been the more openness of the platform, which allows low level access to the hardware. This is a requirement for the application because it has to be able to manage the wireless connection (connect and disconnect to and from the gateways) as well as close control on the camera to ensure syncing between the recorded video and the data stream.

The selected communication network is Wi-Fi because of its wide usage and availability on modern smart devices.

3.2.4 Headquarters server application

In order to allow remote maintenance operations the headquarters server has to support the following features:

- Storage for machine video and data acquisitions
- Storage for firmware upgrades
- API for allowing connections from the smart device
- Web UI for allowing service personnel interaction

To allow easy access to the application, the solution is based on a Web accessible application that can be used through a common browser.

The architecture of the portal is represented in Figure 3.7. The architecture is based on two data sources: a database and a NFS (Network File System) based on a NAS (Network Attached Storage). These two data points are managed by the storage manager component that makes the storage technology invisible to all the other components that need to read and write data, by providing an interface that allows access through defined functions. All the main services that need data access are then connected to the storage manager. The synchronization system manages the

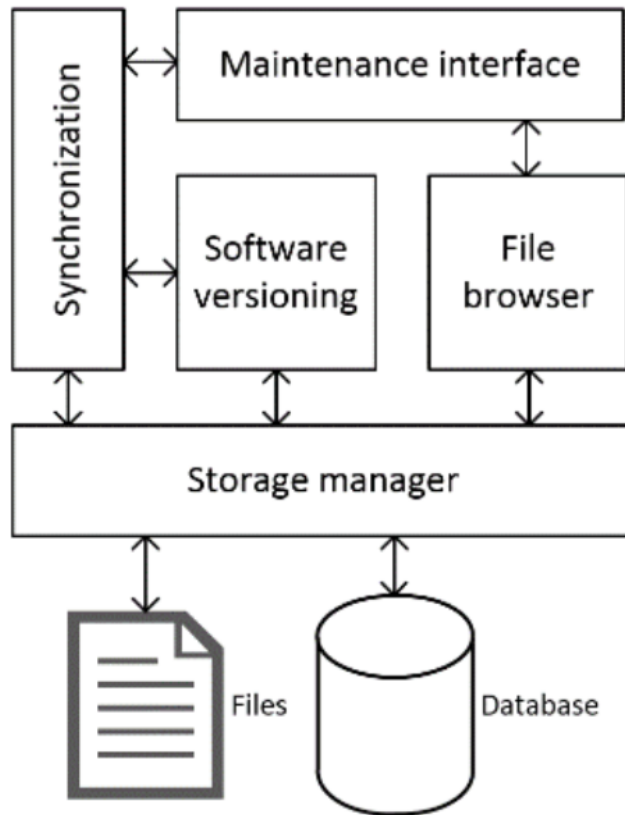


Figure 3.7: The headquarters server application architecture

API used by smart device app for downloads and uploads of files (acquisitions, configurations and software).

Service centre application

The service centre application is a Windows application that is meant for usage by the service trained personnel. Once a package (video and data recorded by the smart device application) is downloaded, this app can open it and visualize its contents. The

main features of the application are:

- Plotting of acquired signals
- Reproduction of recorded video (synchronized with the data plotting)
- Automatic mapping of signal to labels depending on the machine configuration
- Slow motion and sample by sample stepping

The application has been developed for the Windows operating system.

Chapter 4

Implementation and use case

With the purpose of outlining the implementation of the presented system, a use case where the methodology has been applied is provided. The selected use case device is an industrial manipulator, see Figure 4.1, composed of a vertical body, a motor, an arm and a rope linked to the motor at one end, and to an end effector on the other end. The motor is used to lift the load by compensating its weight. With this mechanism the operator can lift and move the load without any effort because all the weight is compensated by the motor power.

The described part is the standard, common part of all the

versions of the device. The part that changes and is available in lots of configurations is the end effector. Depending on the specific application the manipulator is built for, the end effector can have vacuum pumps, grippers, joints, and any other component that is needed for the specific purpose is achieving. This system is suitable for an application of the method described in this thesis because the majority of the automation software is common, regardless of the specific end effector configuration. Thus, each time a new machine is required, the most of the software can be reused, and the only part that needs to be changed is the end effector related software. In the specific configuration of the manipulator where the method has been applied, the end effector is composed by a gripper, buttons for closing and opening the gripper, a sensorized handle for moving the load and a load balancer that moves back and forth for keeping the center of gravity of the load aligned with the rope.

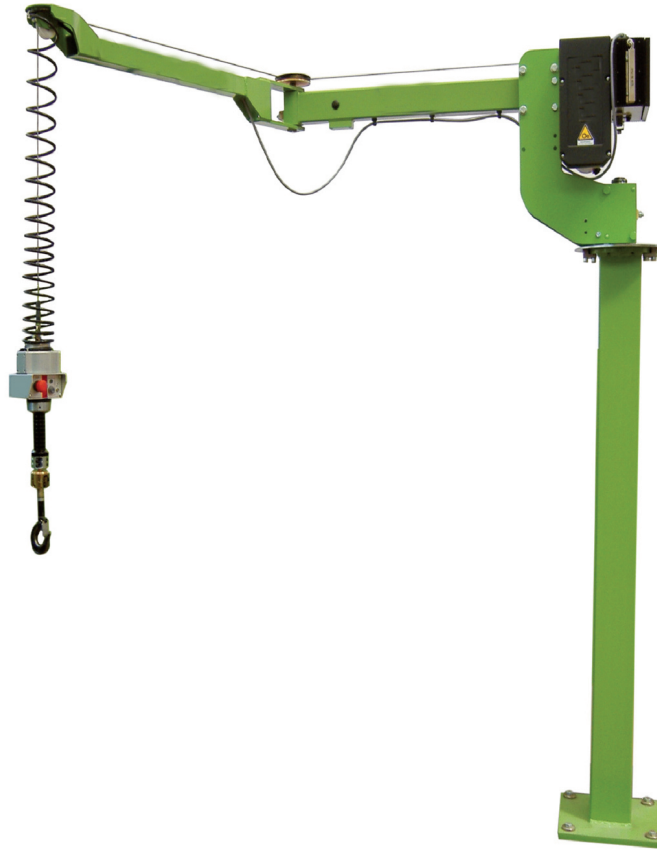


Figure 4.1: The manipulator used in the use case

4.1 Software development

The following sections go through the software development process with references to the introduced manipulator.

4.1.1 Definition and selection of modules

Following the described process the first phase is the definition and selection of modules that compose the machine. Compared to the monolithic approach, where some modules presence would mean just to copy and paste some lines of code from a previous set up, this of course introduces some overhead due to the encapsulation of the features within the blocks, and the definition of interfaces to access the internal functionality. This, on top of having to define the module metadata for every component adds of course additional effort on the first application of this approach, but can save a lot of time later on, whenever any of the defined modules will be reused as this will guarantee to have a properly tested, versioned and reusable module that can be added to the machine software in a similiar way compared to the addition of a mechanical module on the machine structure. Within the use case manipulator the following have been identified:

- *handle*: the handle has sensors for reading the force applied by the operator. It is an input module, the block has only one output interface that provides the read value.
- *load balancer*: the load balancer is basically composed of a gear connected to a rack. When the gear rotates the rack

moves back or forth. It is used for keeping the load center of gravity aligned with the rope. This device can be found in two versions, one with two end switches for detecting if the gear reached the end of the rack, and another one with a potentiometer that gives the absolute position of the gear. Depending on the version mounted on the physical device, a parameter of the module has to be set to switch between the two software implementations in the block. The load balanced used in the real case has two end switches

- *gripper*: the gripper is used to hang the load. It has two end switches for determining if it is fully open or fully closed. The relative module has an input to receive the command for the gripper, and two outputs to provide the value of the two end switches.
- *input button*: the input buttons are used to open and close the gripper. These are just buttons that the operator can press, so they have only one output, the status of the button (pressed / not pressed).
- *load cell*: the load cell is a sensor for reading the load attached to the manipulator. The software module has only one output which gives the value of the attached weight.

- *display*: the display represents the main HMI of the manipulator. The relative software block has only one input, that allows to set the displayed text.
- *lamp*: an output lamp that can be turned on or off. It has only an input to provide the status.
- *motor*: the motor is used for compensating the load hung on the end effector. This module has three inputs: the torque needed for compensating the weight, a maximum speed for lifting, and a maximum speed for dropping (meant for safety reasons).
- *load cell*: the load cell is used for measuring the load of the lifted object. It has basically just one output, the measured weight.

Each module has been described using eXtensible Markup Language (XML) with a scheme defined to contain all the required metadata. The metadata contained is related to versioning, interfaces, documentation and parameters. All these information will be used for HMI and documentation generation. After defining the XML metadata, the real development of the module software takes place.


```
1 <?xml version="1.0" encoding="utf-8"?>
2 <module name="Balancer">
3   <parent_node></parent_node>
4   <version>1.0</version>
5   <description>Balancer of load on the end-effector</description>
6
7   <properties>
8     <property name="with_potentiometer" type="boolean">
9     </property>
10    <property name="max_load_weight" type="enum">
11      <value>40</value>
12      <value>50</value>
13      <value>60</value>
14    </property>
15    <property name="drive" type="enum">
16      <value>electric</value>
17      <value>pneumatic</value>
18    </property>
19  </properties>
20
21  <warnings>
22    <warning linked_property="drive" property_value="electric"
23      symbol="symbols/balancer/electric.jpg" id="balancer_electric"></
24    warning>
25    <warning linked_property="drive" property_value="pneumatic"
26      symbol="symbols/balancer/pneumatic.jpg" id="balancer_pneumatic"><
27    /warning>
28  </warnings>
29
30  <risks>
31    <risk linked_property="max_load_weight" property_value="60" id="
32      balancer_weight60"></risk>
33    <risk linked_property="drive" property_value="all" id="
34      balancer_drive"></risk>
35  </risks>
36
37  <checks>
38    <check linked_property="drive" property_value="electric">
39      <item>attach electric danger sign</item>
40      <item>test the safety of the cables</item>
41    </check>
42    <check linked_property="drive" property_value="pneumatic">
43      <item>test air pressure</item>
44    </check>
45    <check linked_property="max_load_weight" property_value="all">
46      <item>attach load danger sign</item>
47    </check>
48    <check linked_property="with_potentiometer" property_value="true
49      ">
50      <item>check the right reading of middle positions</item>
51    </check>
52  </checks>
```

```

46
47 <io_symbols>
48   <io_symbol type="AOUTPUT" help="command to the balancer motor">
49     balancer_motor_command
50   </io_symbol>
51   <io_symbol type="DINPUT" help="digital input for limit-switch
52     back">
53     limitSwitchB
54   </io_symbol>
55   <io_symbol type="DINPUT" help="digital input for limit-switch
56     forward">
57     limitSwitchF
58   </io_symbol>
59 </io_symbols>
60
61 <app_symbols>
62   <app_symbol type="boolean" connection="AOUTPUT" help="The
63     current position of the balancer">
64     current_position
65   </app_symbol>
66   <app_symbol type="enum" connection="AINPUT" help="The desired
67     position">
68     desired_position
69   </app_symbol>
70 </app_symbols>
71
72 <parameters>
73   <parameter type="enum" linked_property="drive" property_value="
74     electric" var_name="max_I" position_code="20.10.1" value_max="0"
75     value_min="0">
76     <value>1000</value>
77     <value>1300</value>
78     <value>1800</value>
79     <value>2500</value>
80   </parameter>
81   <parameter type="numeric" linked_property="drive" property_value
82     ="all" var_name="max_vel" position_code="20.20.0" value_max="3500
83     " value_min="700">
84   </parameter>
85   <parameter type="graphic" linked_property="with_potentiometer"
86     property_value="true" var_name="SB160" position_code="20.30.0"
87     value_max="0" value_min="0">
88   </parameter>
89 </parameters>
90
91 <documentation id="bar_doc">
92   Lorem ipsum dolor sit amet..
93 </documentation>
94 </module>

```

Listing 4.1: Example of XML Module

Figure 4.4 represents the internal implementation in Simulink of the Balancer block (which XML representation is listed above) as an example. The block reads two end switches from the hardware and then, given the position setpoint, commands the motor to move the rack. The output `CurrentPosition` is updated at each movement to provide the current position value as an output of the block. As defined in the XML description, there is two logical interfaces (`DesiredPosition`, `CurrentPosition`) which can be used by the state machine to interact with the module, and three physical interfaces (`LimitSwitchForward`, `LimitSwitchBack` and `Balancer-MotorCommand`) which are linked to physical interfaces using two *Input Read* and one *Output write* blocks. These blocks are implemented in the firmware, and they are needed to read and write values on the Electronic Control Unit (ECU) pins. The functionalities provided by those blocks is implemented in the so called Base firmware that is the shared basic software that is merged with the machine-specific code at deployment.

4.1.2 Design of the control software

The machine working cycle is implemented using a Stateflow block for implementing the finite state machine (see Figure 4.2). This stateflow block uses signals coming from the blocks to trigger tran-

sitions between states of the state machine, and provides output signals to the blocks depending on its internal state (see Figure 4.3). Matlab Stateflow has been selected as tool for representing state machines because it supports static and run-time checks for cyclic problems and state inconsistencies for validation of the developed working cycle.

Each of the blocks (except for the central one) represented in Figure 4.3 internally implements the control logic for the corresponding mechatronic component like the one described in Figure 4.4. The logical inputs and outputs of the module are connected to the main state machine which handles the machine working cycle. This allow to keep the implementation of the working cycle separate from the modules implementation. The machine working cycle is the piece of software that is most of the times different from machine to machine while the software blocks of the components are not changing between different machines.

4.1.3 Exporting of a model of the system

The process described until now has provided the control algorithm of the manufacturing system, through the design of a finite state machine and the auto-generation of the corresponding source code in Simulink through MATLAB Coder. The remaining part

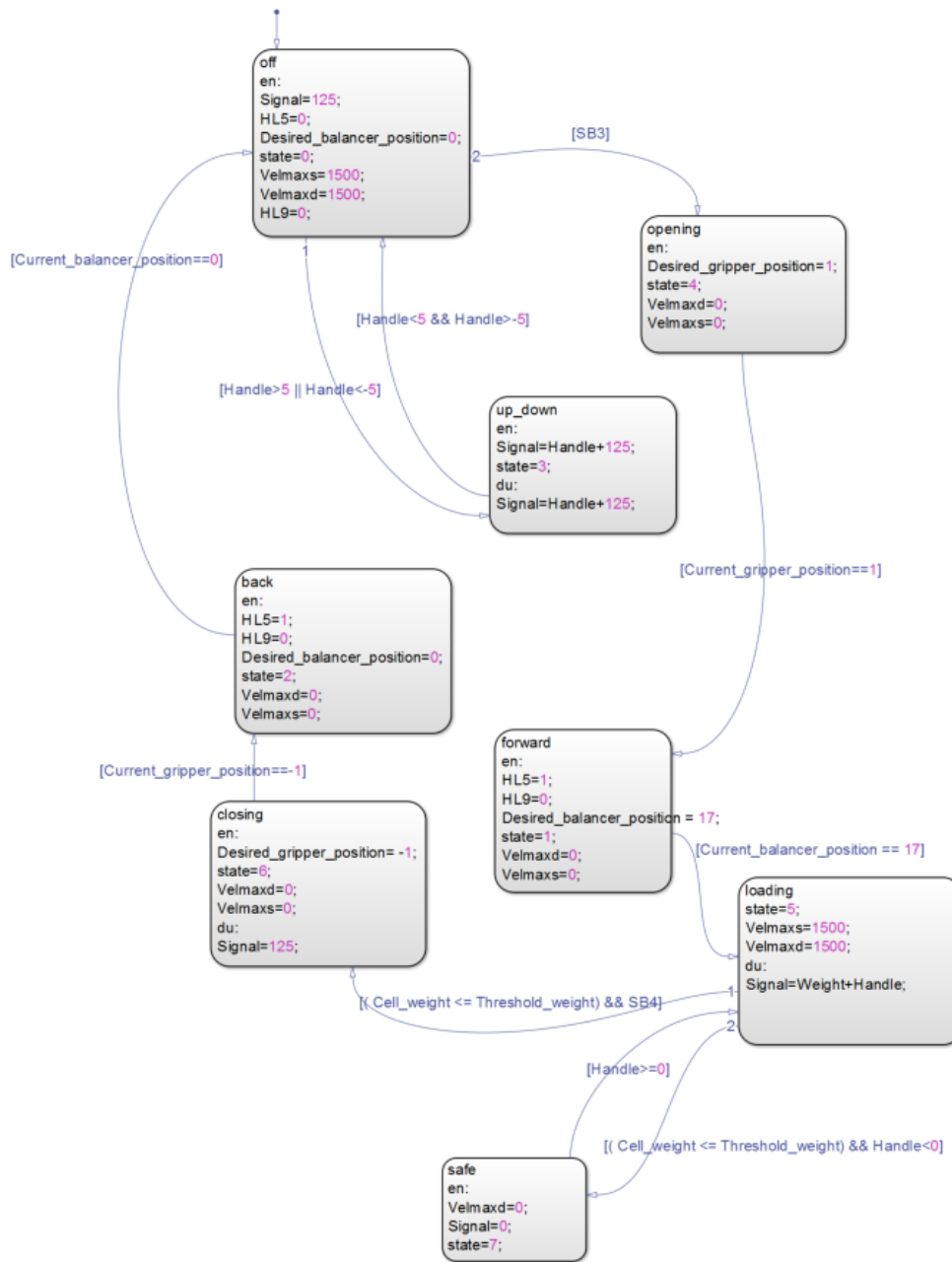


Figure 4.2: The working cycle designed in Stateflow

is the automatic generation of documentation and HMI code.

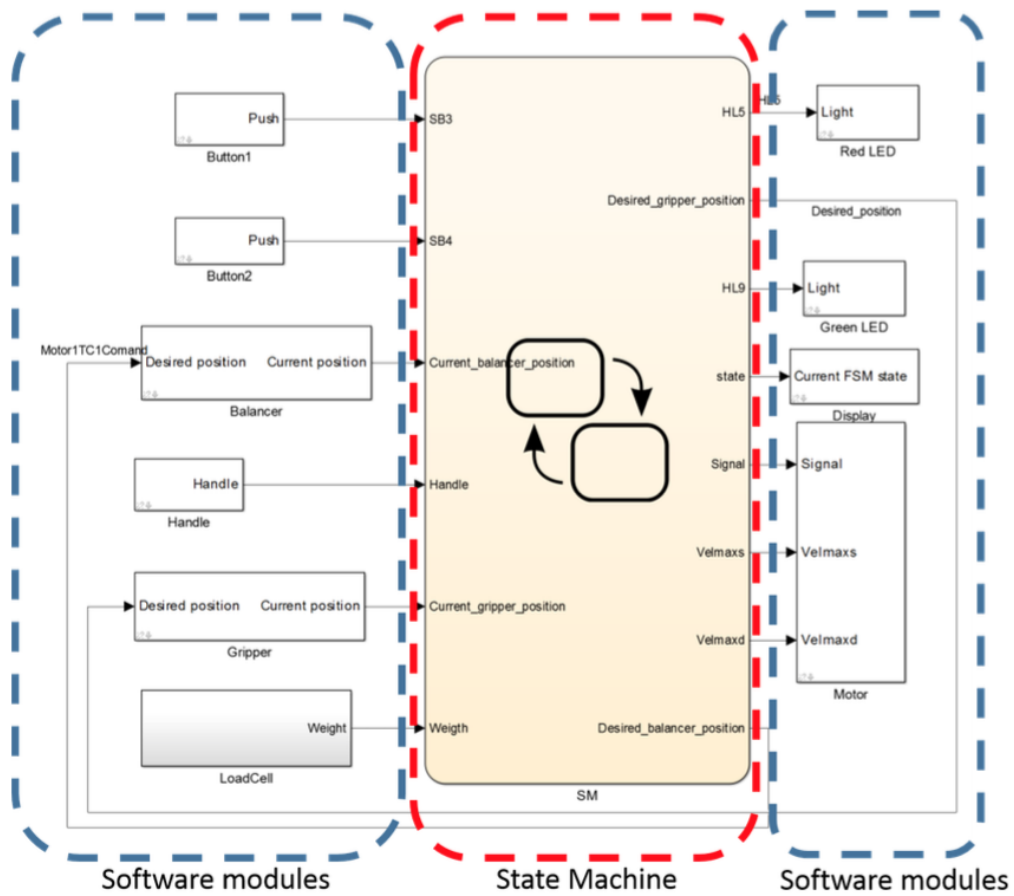


Figure 4.3: The connection between software modules and the state machine

4.1.4 Automatic generation of HMI interface

This step aims at auto-generating HMI interfaces by filling a pre-defined structure with menus and items provided by the loaded modules. It's obviously impossible to automatically generate a completely general HMI, but once the type of machine is defined (like in this example) an interface pattern can be identified, and the place where the module-dependant items have to fit can be

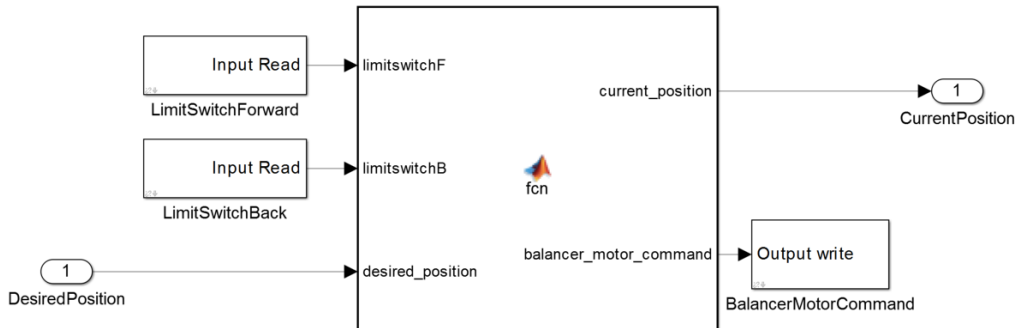



Figure 4.4: The internal implementation of the balancer module

defined. This HMI program is automatically generated by a tool developed for this application. The tool collects all the meta-data of the used modules (the used modules list is taken from the model summary described in the previous section) and prepares the HMI menus composing them with all the items that need to be displayed as specified in the XML definition of the modules. Typically, in this example, the HMI generated contains menus for setting operational parameters, or to display values provided by the machine modules. The current implementation of the HMI generator supports multiple languages if translations are defined in the module XML files. Thanks to this feature, the developer can select which languages should the specific machine support and automatically generate the HMI for just those languages. This is a key feature in the case of this example because the memory

that the controlling unit has available is limited and not all the languages translations can fit in it.

4.1.5 Automatic generation of documentation

Documentation is composed by all the documents that describe the features of the machine, the user manual, the risk assessment document, the menu navigation document (see Figure 4.5 for an example) and the testing checklist. All these documents strictly depend on which modules are loaded on the machine. The script developed specifically for this purpose parses the required modules and generates the doxygen documentation files. Once the Doxygen format is ready, developers can edit its content (it is still a text file) before ultimately generate the PDF version of the document. This could be needed because there can be parts of the documentation which are not modules-related, and have to be modified or created manually. Once the manual editing is done, the Doxygen tool can be invoked to generate the final version of the documents.



Displayable parameters for Base Control Unit				
Display	Definition	Description	Indication troubleshooting	
Init Parameters				
total cycles	number of work cycles done	a cycle is counted when you pick and place a load		
Signal	power provided to the lift motor	the value means 1A each 30	only the load cell needs of 125 to keep itself stable	
Balancer Parameters				
max current	max current the motor can provide	the value set is in mA	with a heavy load, a low level of current can be insufficient	
max speed	max speed the balancer can move		set a value from 700 to 3500	
bar position	middle position of the balancer	this parameter allows a manual management of horizontal load position	8 different position available	

Figure 4.5: An example of the generated documentation for menu navigation

4.2 Remote maintenance

The same use case, has been used to apply the remote maintenance approach. As stated in the previous section, the machine where this method has been applied to is an industrial manipulator. In this section some details about the technical implementation of communication on the device are described with the purpose of explaining how the remote maintenance system can be applied to an existing machine. After the manipulator architecture is introduced, the connection to the gateway is explained and a closer view to the current implementation for this device described.

4.2.1 The manipulator architecture

The machine is composed of three electronic boards that communicate through a CAN backbone. This makes the device perfect for this application because the gateway can be installed on the CAN backbone and it is allowed to read all the data that runs inside it. The three boards of the machine are:

- **Power supply board**

It manages the power supply for the machine, rectifying the voltage network to get 24 V DC for the electronic components and 65 V DC for the main motor of the machine

- **Motor control board**

It manages, controls and monitors the main motor of the machine, in order to perform the optimal movement depending on the speed set-point generated by the control board. This board is also necessary to control the safety devices of the machine;

- **Command board**

This board implements all the algorithms to implement the HMI and controls all machine operations. In particular the main activity is to acquire from the operator the desired movements and speeds in order to generate the speed set-

point.

4.2.2 The gateway

The gateway is required to route information coming from the machine fieldbus to a wireless network; therefore two interfaces are needed. The interface for the machine's fieldbus obviously depends on the bus that is available on the machine. The connection for the wireless network instead is based on the standard IEEE 802.11b/g protocol. Other communication protocols have been considered, such as Bluetooth, but that would limit the number of suitable smart devices that can be used for communication. iOS devices in fact have limitations imposed by the Apple on Bluetooth connections that would require a certification of the gateway (for more details on these limitations see Apple MFi Program). Using Wi-Fi, the gateway is compatible with a wide range of available smart- phones, guaranteeing a high standard of performance in the communication.

- Analog input (0-3.3V)
- Digital input (24V)
- Digital output (24V)
- Serial communication interface (RS-232)

The selected module for the wireless connection is a Wi-Fi Roving Networks RN-131G/C, directly connected to the controller through a serial port. This module supports multiple working modes:

- **Infrastructure**

The module is connected to an existing Wi-Fi network

- **Ad Hoc**

A point-to-point communication channel can be created (between the module and the smartphone)

- **Soft-AP**

The behavior of the radio module is equivalent to an access point (with a maximum number of seven clients)

The configuration used in this work is Soft-AP: with this configuration, each gateway exposes a network where the smart device can connect to debug the machine.

The data acquired from the machine is necessary to determine the real functioning status and, consequently, to define the maintenance interventions and how to manage these operations remotely. The logs collected can be sent to the manufacturer for further analysis by skilled personnel who will determine if the machine is in good shape or if something is not working correctly. With this

purpose, the main data acquired in this applicative example are related to:

- Motor current
- Motor voltage
- Motor speed
- Operator commands
- Handling times
- General I/O
- Alarms and warning messages

Currents and voltages can be analyzed to see if the electrical components are functioning properly. Through these signals and the speed, it is possible to diagnose the motor - the main actuator of the machine - to determine its condition and whether it is malfunctioning and, if so, what type of failure is present. I/O signals and alarms can be used for understanding if the control logic is working properly and if the status of the electromechanical components is good. The movement speed and the handling time can be used to detect problems in the motion system, especially regarding the lift cables, the system of pulleys and the grasp system.

Once the gateway is turned on, it announces the SSID of the wireless access point, and waits for a connection from a mobile internet device. The smartphone can then see the wireless network available and connect to it. Once the wireless connection is established, the smartphone opens a TELNET socket to the gateway, which waits for commands.

Commands that the gateway accepts are two simple commands, one for starting the transmission of the data stream and one for stopping it:

- **gotp**: Starts the transmission of the data stream acquired from the machine CAN from the gateway.
- **stoptp**: Stops the transmission of the data stream.

The structure of the data packet sent from the gateway is represented in Figure 4.6. This basic ASCII protocol is enough for the specific application this demo is being applied on.

The data packet is a 40 bytes packet with the structure defined in Figure 4.7:

All the variables are sent one after the other, with two initial bytes set to 0xF0 to determine the beginning of the message. The end of the message is represented by two bytes set to 0x2A, followed by the checksum of the message itself for validation and

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
240	240	Var. 1		Var. 2		Var. 3		Var. 4	
Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Var. 5		Var. 6		Var. 7		Var. 8		Var. 9	
Byte 20	Byte 21	Byte 22	Byte 23	Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29
Var. 10		Var. 11		Var. 12		Var. 13		Var. 14	
Byte 30	Byte 31	Byte 32	Byte 33	Byte 34	Byte 35	Byte 36	Byte 37	Byte 38	Byte 39
Var. 15		Var. 16		42	42	CRC - Var.		15	15

Figure 4.6: Message protocol structure

from other two bytes set to 0x0F to determine the end of the packet.

4.2.3 Smartphone app

The layout of the smart device application is represented in Figure 4.8.

The application is Android-based written in Java: in the Android architecture each screen of the application is a so called *activity*. Figure 4.8 depicts the flow between the application activities with the typical workflow as follows

- **Main screen** The main screen of the application, where the menu with all the possible actions is presented. The user here can select the option to connect to a machine.
- **Options** The options section allow the user to configure the

Var 01: packetCounter	Var 02: Mess_UdC_Feeder
Var 03: Commands	Var 04: Deltal
Var 05: DeltaVelMax	Var 06: Signal
Var 07: VelMaxD	Var 08: VelMaxS
Var 09: CAN_Alarm	Var 10: I2TMot
Var 11: Imot	Var 12: Mess_Feeder_UdC
Var 13: VelMot	Var 14: DigitalInputTC1
Var 15: Motor1TCStatus	Var 15: Motor2TCStatus

Figure 4.7: The data packet structure

quality of the video and its framerate. This is useful on older devices that do not have enough computational power to record high-quality video while receiving data or when the network connectivity is not reliable for sending large files. Other options that can be configured in this view are the server address for the upload and the wireless password for the gateway.

- **Machine selection** From this screen, the user can select which machine he/she wants to connect to. The list of machines is generated by scanning wireless Soft-AP gateways and listing the devices found. Each wireless SSID is called

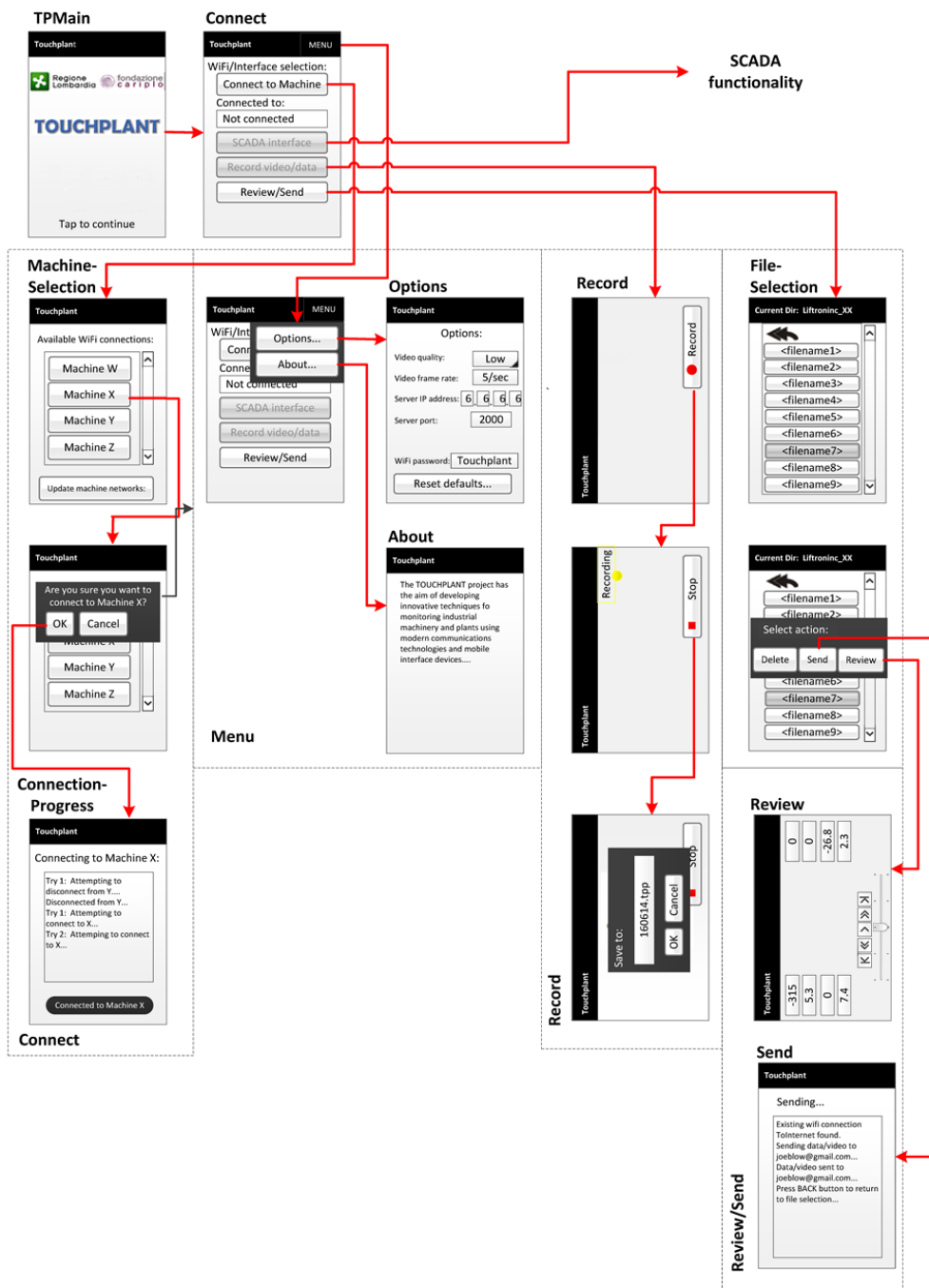


Figure 4.8: The architecture of the smart device application

with the serial number of the machine. Once the user selects the device, the connection is made and maintenance operations are enabled.

- **Machine connection** Once the connection is started, the app shows on the screen the log of the connection attempt it is performing. Once the connection is completed a message is displayed and the user can go back to the main screen.
- **Recording** The user can then select the option to start a new recording acquiring data from the machine is connected to. Once the acquisition is started, the user can frame the operator while interacting with the machine, and stopping the recording when is completed.
- **Save** After the completion of the acquisition, the user is requested to enter a name for saving the video and data collected. The package is saved as a compressed file on the smart device internal storage.
- **Review** After saving the package, the user can review it by reproducing the video on the smart device screen. An overlay is displayed with some of the data collected. Touching the items on the overlay, the user can hide and show other data previously acquired from the gateway.

- **Sending** When the user is happy with the content of the recording, he can proceed sending it to the manufacturer service center.

The remote transmission of video and operational data is currently sent via e-mail with a package attached. The app automatically disconnects from the gateway when the sending is requested to allow the smartphone to connect to another Wi-Fi connection that allows access to the Internet. The format of the transmitted package consists in a zip archive, containing:

- An mp4 video: the recorded audio/video using the camera compressed with an mp4 codec.
- A TXT file: the ASCII text file containing all the data collected from the gateway.

The text file has three header lines that precede the lines of acquired data. The header is structured as follows:

- **First row** Serial number of the machinery, date and time of recording (E.g. “SN:2015245455;2015-04-15;15:30:25”)
- **Second row** Name of the variables as semicolon separated list (E.g “TimeStamp;PcktNmbr;MessUDCFeeder;Commands;DeltaI;DeltaVelMax;Signal;VelMaxD;VelMaxS;CANAlarm;

I2TMot;Current;MessFeederUDC;MotSpeed;DITC1;
M1TC1St.;M2TC1St.;Mass;Speed;Position;Mass2;
Speed2;Position2;SpeedRef;HumanInput;MachineState;
SpeedRef2;HumanInput2;MachineState2;-;-")

- **Third row** Units of measure of the variables listed in the second row (E.g. "(sec);(-);(-);(-);(A);(rpm);(-);(rpm);(rpm);(-);(-);(A);(-);(rpm);(-);(-);(-);(kg);(rpm);(-);(kg);(rpm);(-);(rpm);(-);(-);(rpm);(-);(-);(-);(-);(-)")

4.2.4 Windows app

In order to diagnose, debug and maintain a remote machine, a PC application has been developed to review and analyze the transmitted information. Maintenance services usually work with standard PCs, so this program was written for use in the Windows environment. The application consists of a simple one-form software (see Figure 4.9) developed in Visual C# using .NET framework version 4. This version has been preferred in comparison to the latest available because it would allow wider support and coverage for currently available industrial PCs, allowing the software to run on Windows XP.

CHAPTER 4. IMPLEMENTATION AND USE CASE

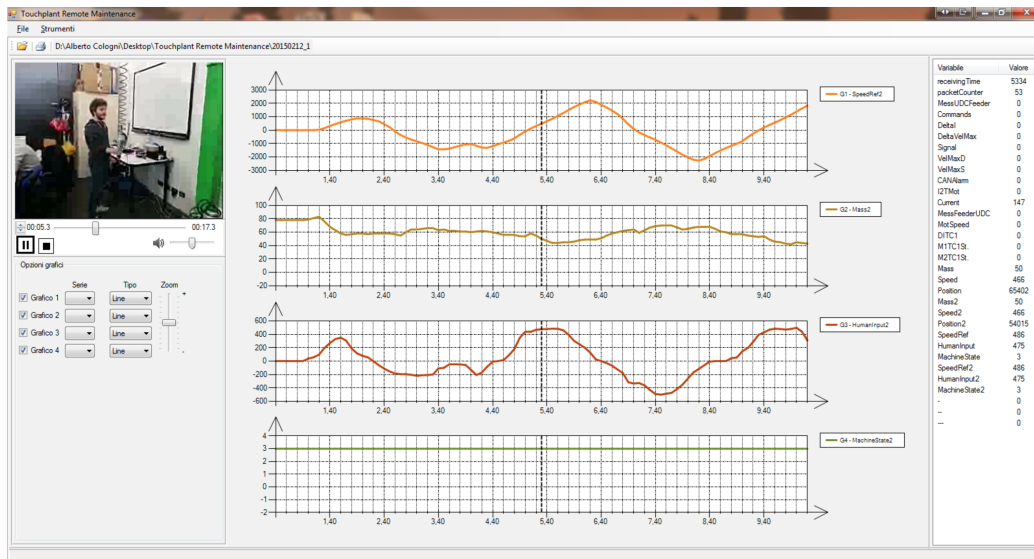


Figure 4.9: The developed Windows application

Starting from the file menu it is possible to open the received files (using a local path or a network URL. Once the file is open it is possible to watch the acquired video while the acquired signals are plotted on a time chart.

The user interface is represented in Figure 4.10

Various sections are:

1. Menu bar
2. Tools bar
3. Video player and controls
4. Plotting options: (number of plots to display, which variables to plot, zoom settings)

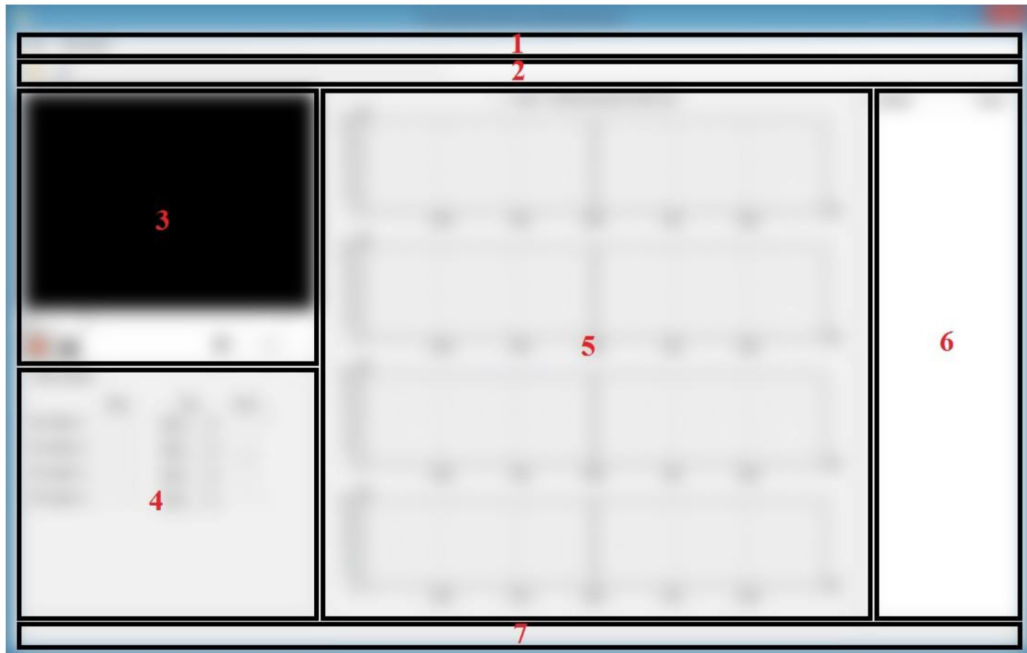


Figure 4.10: The sections of the Windows application

5. Plots
6. Instantaneous variables values
7. Status bar

The plotting area can be populated with 1-4 plots, which can be activated or hidden using the plotting options. Hiding some plots will increase the size of the remaining ones automatically. Every plot can show up to 6 variables, thus the entire application allows to plot 24 variables on the 4 plots.

Once a package is opened in the application, a technician can analyze the signals synchronized with the video stream. The video

player timeline is linked with the plots timeline, allowing the engineer to see the signals status for every operation performed in the video. This allows a remote support engineer to understand what is happening on the machine control unit, while having an immediate idea of how the machine has been used, and what was the context where the machine was working. In the case of semi-automated machines where the interaction with the operator is a key factor, having an idea of how the operator was interacting with the machine is crucial to identify issues and areas of improvement. This is mostly the case where the interaction of the human with the machine is not purely achieved through digital inputs, but is also a physical interaction, where the human alters the environment that the machine expects around it. In this case the user is part of the machine movement and interaction and all this working context is interesting at the debugging stage.

If further investigation with more advanced tools and algorithms is required, the Windows application provides a feature to export all the data variables to a MATLAB “.mat” file, which is a data format that can be imported within the Mathworks MATLAB tool. This tool has several features for data analysis and great plotting abilities.

The main features of the diagnostic program are:

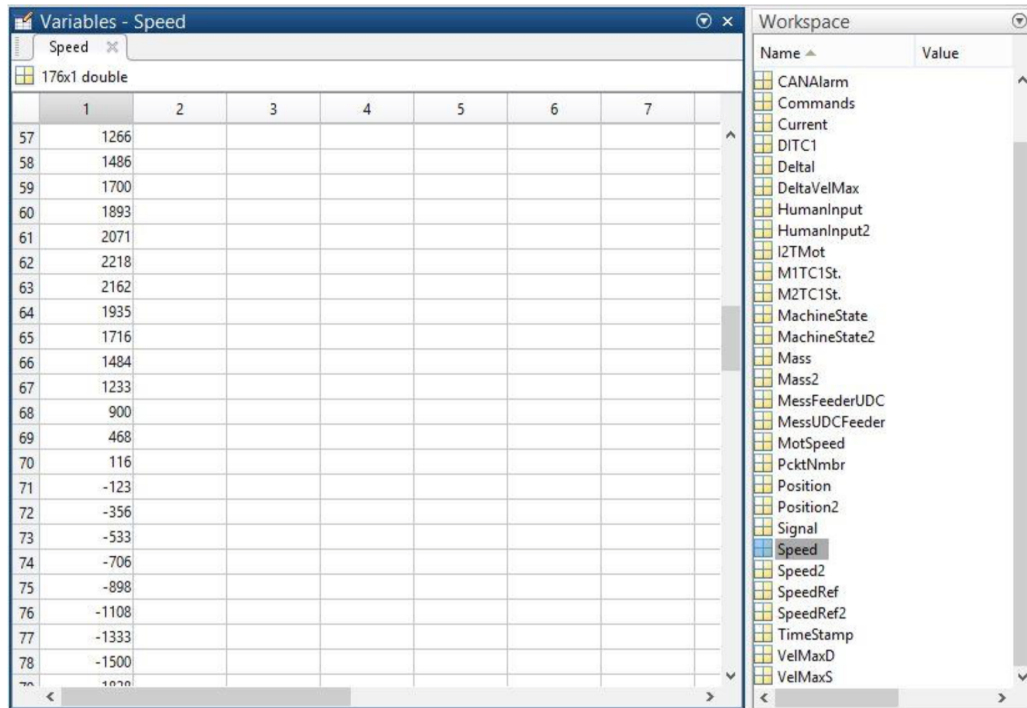


Figure 4.11: The MATLAB exported file

- Manage the video stream as a typical video player (play, pause, forward, rewind, stop, etc.)
- Define and change which signals are plotted on the four available axes, with a maximum of 6 signals per plot
- Move the video display and use drag and drop to change the signals plotted
- Verify all the acquired signals

In order to customize the acquired signals for each machine, before each connection to a machine, a JSON configuration file is

loaded from the service center server. With this arrangement, it is also possible to change the number of acquired signals remotely and assign, for each one, a meaningful name.

4.2.5 Synchronization

In order for this system to be useful, a good synchronization between data and video stream is required. Having delays between such two streams in fact can prevent the technician to identify issues in the signals running on the machine, or identifying causes for specific conditions. For example, if unexpected data is identified in the data stream, this may be due to some action performed by the user, which can be identified only if the video stream is synchronized with the data itself.

Given the importance of the synchronization of these two streams, an investigation has been carried out to verify to what extent the presented solution is robust.

The details about acquisition for the streams are:

- **Audio/video stream** acquired with a sample rate of 30 Hz, with the support of the smartphone digital camera and microphone.
- **Data stream** acquired with a sample rate of 10 Hz, through

a Wi-Fi connection between the Gateway and the smart-phone.

With the objective of minimizing the transmission time between the mobile device and the gateway, the connection is always established as a single-hop connection via a wireless network, with the Gateway acting as an access point. In this way, the two devices are always directly connected one another.

To make sure to not introduce delays or misalignments between the two streams due to the architecture of the application itself, some expedients have been used.

When the user wants to start the acquisition, a socket connection to the gateway from the application is opened. Based on the operating system behavior, there can be an undefined delay between the request by the user to enable the camera, and the actual readiness of the peripheral, depending on the availability of the resource, and on other tasks running at that time. For this reason, before sending the start message to the gateway which will trigger the acquisition of data, the camera access is requested to the operating system. Once the response from the operating system is received, and thus the camera is ready, the start message is written on the gateway socket.

Two different threads, take care of handling the two different

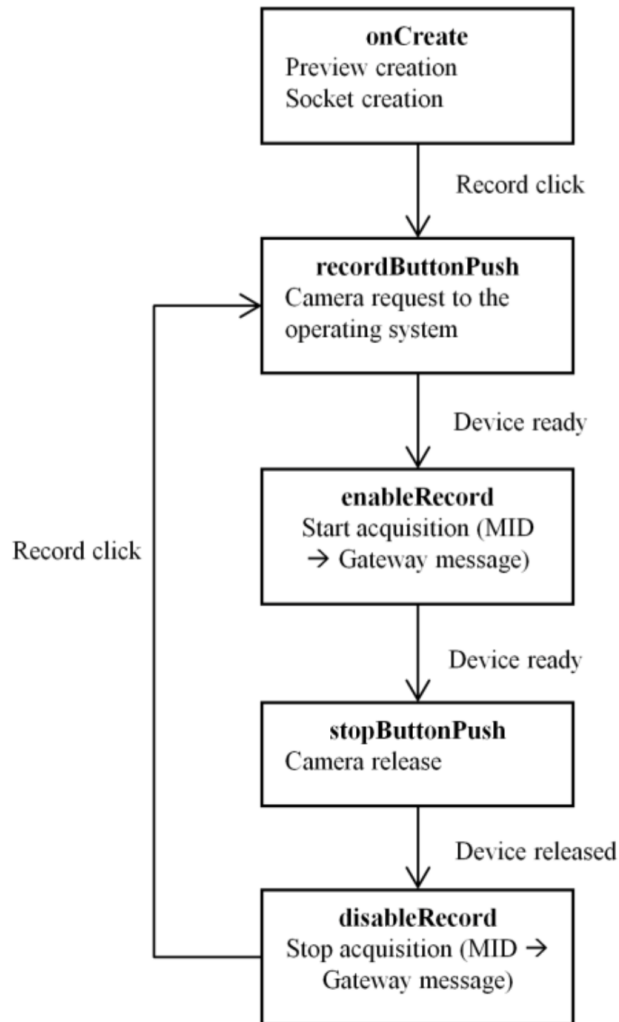


Figure 4.12: The phases of a recording

streams separately. The video stream is acquired by the video manager, that writes it on an mp4 file on the device, while the data-stream manager writes an ASCII file containing the received data. Every time an entry is written within the data file, informa-

tion about elapsed time for that entry is stored as well, to allow synchronization when reproducing the streams together.

The gateway continuously sends packets until a stop message is received by the mobile device. A scheme of the transmission operation is depicted in Figure 4.13.

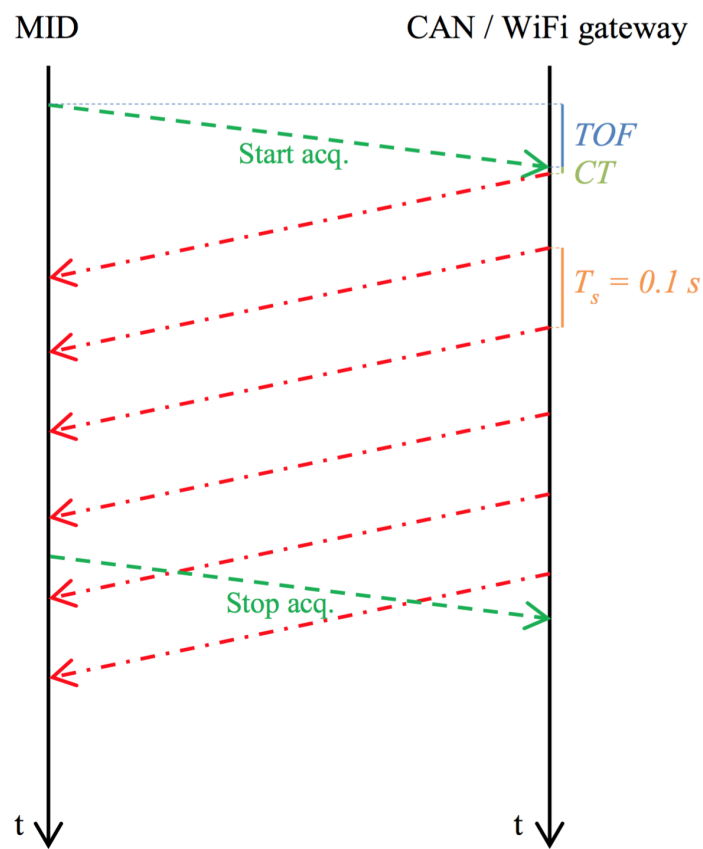


Figure 4.13: Transmission scheme

To be able to evaluate the performances of the synchronization process, some terms have to be defined:

- **TOF (Time of Flight)** This is the time that a packet requires to move from one point to another within a network. In our case, it represents the time a data packet takes for being transferred from the gateway to the device. This Time of Flight is considered constant and symmetric after validating this assumption experimentally.
- **CT (Computation Time)** It represents the time for the elaboration of the received package by the gateway.
- **TS (Sample Time)** This is the time between two consecutive samples.

The experiment carried out aims at computing the value of the sum of TOF and CT by sending a package from the mobile device to the Gateway onboard the machine and configuring it to respond to each received package with a simple confirmation package. This operation is repeated several times and all the timing results logged to derive statistical data from them. The duration of CT has been measured by changing the value of a digital output at the beginning and end of the evaluation of the acquisition algorithm and measured with external equipment. A scheme of the test set up is represented in Figure 4.14.

Results of the experiment are reported in Table 4.1

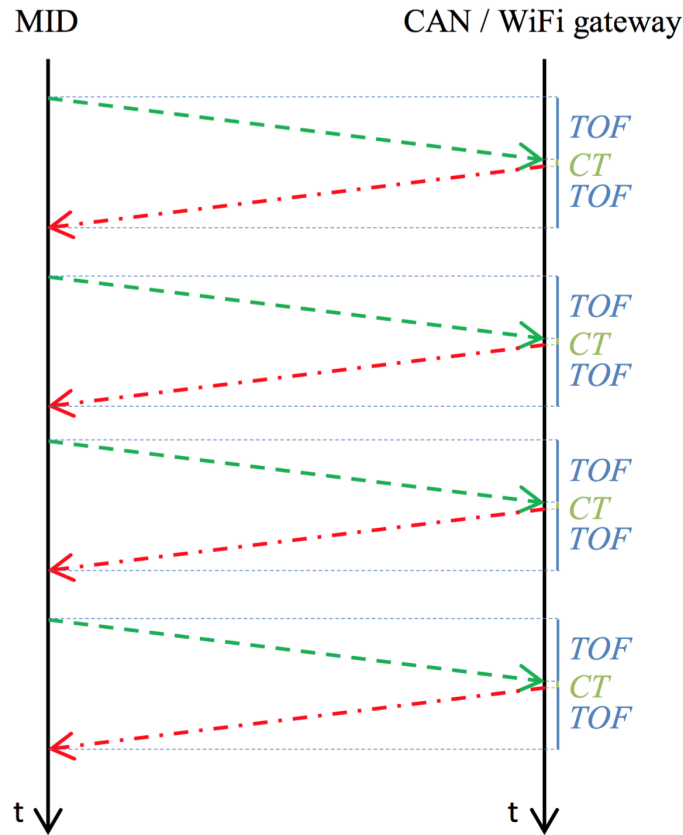


Figure 4.14: Test scheme

	Average value [ms]	Standard deviation [ms]
TOF	4.51	1.7
CT	0.73	0.012

Table 4.1: Test results

The mean Time of Flight for the transmission is 4.51 ms, while the average for Computation Time is 0.73 ms. Considered that the sampling rate T_s is 0.1 s, the Time Of Flight + Computation Time is less than 6% which makes this delay neglectable.

Chapter 5

Conclusions and future work

In this thesis, an automation software development method and an integrated remote maintenance platform for semi-automated machines have been presented. Chapter 2 has given a review of the state of the art on software development in automation and on systems engineering techniques in general. Additionally, current status of remote maintenance and e-maintenance literature is introduced. Chapter 3 explained the features of the proposed approach for modularization of the automation software, and for remote maintenance using mobile internet devices. The main objectives of the proposed approaches are:

- Decompose mechatronic systems into modules, encouraging reusability and reducing costs in terms of money and time.
- Provide a “user friendly” (possibly graphical) environment for control software design and development.
- Allow automatic generation of control algorithm source code, starting from the model of the system.
- Allow automatic generation of Human Machine Interface source code and enable automatic writing of product documentation, such as user’s manual.
- Leverage widely available mobile internet devices for acquiring debugging data from machines, while recording video for remote technicians.
- Retrofit old machinery for supporting transmission of data to mobile internet devices.
- Allow technicians to review the recorded packages using a player which enables the contemporaneous visualization of video and data.

The approaches presented in Chapter 3, are then applied to a real case in Chapter 4. The selected machinery is an industrial

manipulator which constitutes a good test bench given its customizability and its interaction with humans during its operation.

5.1 Limits and strenghts

A rapid excursus of the limits and strengths of both the approaches presented as a contribution of this thesis are detailed in this section.

5.1.1 Automation software development method: limits and strenghts

The presented solution for organizing software into modules and optimize the customization and the automatic generation of software and other artifacts proved to be a valid solution during its implementation on the test bench manipulator. The main area of improvement of this work is supporting automated verification and validation of the developed software state machine. Having the state machine already developed within Simulink environment allows to perform these activities, but this support has not been investigated as part of this work. The potential expressed in the metadata associated to every module, opens horizons to integrate even more data. So far, just documentation and risk information

has been associated with every module, while in the future the system has the potential to integrate further information, like:

- **Modules pricing** Which would allow the sales office to formalize offers faster for customers having already an idea of the hardware costs
- **Testing/assembling duration** Which would allow to provide and schedule easily the duration of the assembly of every machine
- **Stock quantities** Which would allow the production office to be aware of how many modules are available, and schedule the production accordingly

5.1.2 Remote maintenance solution for semi-automated machines: limits and strengths

The proposed solution for remote maintenance enables the simultaneous acquisition of multiple streams from a running machine while an operator is interacting with it. Collected data is:

- Video recording of the operator interacting with the machinery

- Field bus data streams, acquired directly from the field bus on the machine

The current implementation of the platform allows the acquisition of data up to 10Hz. By using this sample rate, given the nature of the acquired data there are no issues regarding the synchronization between video and data streams and the system can handle it with good performances. In applicative cases where a higher sample rate is needed, issues regarding the synchronization between data and video stream can potentially show up.

The other main limitation of the platform, is the non-real-time nature of the operating systems on the mobile internet devices. The performance of the acquisition demonstrated to be satisfying in all the cases but considering that the operating system running on the devices is a multi tasking environment, there can be cases where an overloading of the CPU from other processes can slow down the acquisition and storage of the synchronized stream, potentially causing synchronization issues. To address this case, a performance monitor can be implemented on the smartphone app, to validate if the load of the CPU is low enough for allowing the synchronization. Another possible solution would be to validate the input buffer of the TELNET connection. If the buffer contains more than 1 packet in the queue, then it means that the

smartphone CPU is not able to keep up with the rate of the incoming data, which can trigger an alarm to inform the user that the video/data is recording are going to be out of sync. This can suggest the user to close other running processes to free up enough CPU time for handling these operations.

As opposed to the identified limits, the main strength of the proposed solution is that it allows the implementation of an integrated remote maintenance system based on devices widely available on the market, which even if not optimal, provide enough performance to fulfill the requirements of several systems like the one presented in the applicative case in this thesis. Several improvements can be made on this implementation, leveraging for example a real time operating system for the mobile internet device, which can potentially guarantee the processing of data, but that would require the mobile internet devices to be dedicated to this specific purpose instead of allowing a quick acquisition on multi purpose devices that are widely spread and available in everyday life.

Bibliography

- [1] Thomas Bangemann, Eric Garcia, Christophe Lang, Xavier Rebeuf, Jacek Szimanski, Jean-Pierre Thomesse, and Mario Thron. Proteus-a european initiative for e-maintenance platform development. In *9th IEEE International Conference on Emerging Technologies and Factory Automation*, 2003.
- [2] Thomas Bangemann, Xavier Rebeuf, Denis Reboul, Andreas Schulze, Jacek Szymanski, Jean-Pierre Thomesse, Mario Thron, and Nouredine Zerhouni. Proteus - creating distributed maintenance systems through an integration platform. *Computers in Industry*, 57(6):539–551, 2006.
- [3] Don Batory. Feature models, grammars, and propositional formulas. In *International Conference on Software Product Lines*, pages 7–20. Springer, 2005.
- [4] AA Alvarez Cabrera, MJ Foeken, OA Tekin, K Woestenenk,

- MS Erden, B De Schutter, MJL Van Tooren, R Babuška, FJAM Van Houten, and T Tomiyama. Towards automation of control software: A review of challenges in mechatronic design. *Mechatronics*, 20(8):876–886, 2010.
- [5] Jaime Campos, Erkki Jantunen, and Om Prakash. A web and mobile device architecture for mobile e-maintenance. *The International Journal of Advanced Manufacturing Technology*, 45(1-2):71, 2009.
- [6] Paul Clements and Linda Northrop. Software product lines. *Product Line systems Program*, 2003.
- [7] Cristian Colace, Luca Fumagalli, Simone Pala, Marco Macchi, Nelson R Matarazzo, and Maurizio Rondi. Implementation of a condition monitoring system on an electric arc furnace through a risk-based methodology. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 229(4):327–342, 2015.
- [8] dSPACE. dspace, website. <https://www.dspace.com>, 2016.
- [9] Christos Emmanouilidis, Jayantha P Liyanage, and Erkki Jantunen. Mobile solutions for engineering asset and maintenance management. *Journal of Quality in Maintenance Engineering*, 15(1):92–105, 2009.

- [10] Petter Falkman, Erik Helander, and Mikael Andersson. Automatic generation: A way of ensuring plc and hmi standards. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–4. IEEE, 2011.
- [11] Guillaume Finance. Sysml modelling language explained. *OMGSysML.com*, www.omgsysml.org/SysML_Modelling_Language_explained-finance.pdf, 2010.
- [12] Georg Frey. Automatic implementation of petri net based control algorithms on plc. In *American Control Conference, 2000. Proceedings of the 2000*, volume 4, pages 2819–2823. IEEE, 2000.
- [13] Georg Frey and Lothar Litz. Formal methods in plc programming. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2431–2436. IEEE, 2000.
- [14] Luca Fumagalli and Marco Macchi. Integrating maintenance within the production process through a flexible e-maintenance platform. *IFAC-PapersOnLine*, 48(3):1457–1462, 2015.
- [15] H-M Hanisch, J Thieme, Arndt Luder, and O Wienhold.

- Modeling of plc behavior by means of timed net condition/event systems. In *Emerging Technologies and Factory Automation Proceedings, 1997. ETFA '97., 1997 6th International Conference on*, pages 391–396. IEEE, 1997.
- [16] Martin Hirsch. *Systematic design of distributed industrial manufacturing control systems*, volume 6. Logos Verlag Berlin GmbH, 2010.
- [17] Martin Hirsch and Hans-Michael Hanisch. Systemspezifikation mit sysml für eine fertigungstechnische laboranlage. *Fachtagung zum Entwurf komplexer Automatisierungssysteme (EKA 08)*, pages 23–34, 2008.
- [18] Min-Hsiung Hung, Kuan-Yii Chen, Rui-Wen Ho, and Fan-Tien Cheng. Development of an e-diagnostics/maintenance framework for semiconductor factories with security considerations. *Advanced Engineering Informatics*, 17(3):165–178, 2003.
- [19] C Ie. Programmable controllers—part 3: Programming languages. 2003.
- [20] IEC 62264-1:2013. Enterprise-control system integration. Standard, International Organization for Standardization, March 2013.

- [21] INCOSE. IncoSE, what is systems engineering. <http://www.incose.org/AboutSE/WhatIsSE>, 2016.
- [22] (IEC) International Electro-technical Commission. International standard iec61499, function blocks. *IEC Jan. 2005 Edition 1.0.*, <http://www.iec.ch/>, (part 1-4).
- [23] ISO 13374-3:2012. Condition monitoring and diagnostics of machines – Data processing, communication and presentation. Standard, International Organization for Standardization, March 2012.
- [24] ISO 15745-3:2003. Industrial automation systems and integration – Open systems application integration framework – Part 3: Reference description for IEC 61158-based control systems. Standard, International Organization for Standardization, March 2003.
- [25] Benoît Iung, Gérard Morel, and JB Leger. Proactive maintenance strategy for harbour crane operation improvement. *Robotica*, 21(03):313–324, 2003.
- [26] Benoît Iung, Eric Levrat, Adolfo Crespo Marquez, and Heinz Erbe. Conceptual framework for e-maintenance: Illustration by e-maintenance technologies and platforms. *Annual Reviews in Control*, 33(2):220–229, 2009.

- [27] Kouroush Jenab and Saeed Zolfaghari. A virtual collaborative maintenance architecture for manufacturing enterprises. *Journal of Intelligent Manufacturing*, 19(6):763–771, 2008.
- [28] Charles W Krueger. Introduction to the emerging practice of software product line development. *Methods and Tools*, 14(3):3–15, 2006.
- [29] Jay Lee. Strategy and challenges on remote diagnostics and maintenance for manufacturing equipment. In *Reliability and Maintainability Symposium. 1997 Proceedings, Annual*, pages 368–370. IEEE, 1997.
- [30] Jay Lee. A framework for web-enabled e-maintenance systems. In *Environmentally Conscious Design and Inverse Manufacturing, 2001. Proceedings EcoDesign 2001: Second International Symposium on*, pages 450–459. IEEE, 2001.
- [31] Kang B Lee and Richard D Schneeman. Internet-based distributed measurement and control applications. *IEEE Instrumentation & Measurement Magazine*, 2(2):23–27, 1999.
- [32] Paulo Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, 2009.

- [33] Eric Levrat, Benoit Iung, and Adolfo Crespo Marquez. E-maintenance: review and conceptual framework. *Production Planning & Control*, 19(4):408–429, 2008.
- [34] Robert Lewis. *Modelling control systems using IEC 61499: Applying function blocks to distributed systems*. Number 59. Iet, 2001.
- [35] Thomas Mertke and Georg Frey. Formal verification of plc programs generated from signal interpreted petri nets. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 4, pages 2700–2705. IEEE, 2001.
- [36] MIMOSA. Mimoso, website. <http://www.mimosa.org/>, 2016.
- [37] R Keith Mobley. *An introduction to predictive maintenance*. Butterworth-Heinemann, 2002.
- [38] László Monostori, József Váncza, and Soundar RT Kumar. Agent-based systems for manufacturing. *CIRP Annals-Manufacturing Technology*, 55(2):697–720, 2006.
- [39] Gasper Music, Dejan Gradisar, and Drago Matko. Iec 61131-3 compliant control code generation from discrete event models. In *Proceedings of the 2005 IEEE International Sympo-*

- sium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.*, pages 346–351. IEEE, 2005.
- [40] W Ramus and J Neroda. E-diagnostics: the value proposition story. *Semiconductor International*, 7(1):2003, 2003.
- [41] Wolfgang Reisig. *Petri nets: an introduction*, volume 4. Springer Science & Business Media, 2012.
- [42] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.
- [43] P Sangregorio, AL Cologni, A Piccinini, A Scarpellini, and F Previdi. A method for automation software design of mechatronic systems in manufacturing. *IFAC-PapersOnLine*, 48(3):936–941, 2015.
- [44] Paolo Sangregorio, Alberto Luigi Cologni, Franklin Caleb Owen, and Fabio Previdi. An integrated system for supporting remote maintenance services. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–4. IEEE, 2015.
- [45] Pierre-Yves Schobbens, Patrick Heymans, and Jean-

- Christophe Trigaux. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering, 14th IEEE international conference*, pages 139–148. IEEE, 2006.
- [46] Cristian Secchi, Marcello Bonfé, Cesare Fantuzzi, Roberto Borsari, and Davide Borghi. Object-oriented modeling of complex mechatronic components for the manufacturing industry. *IEEE/ASME Transactions on Mechatronics*, 12(6): 696–702, 2007.
- [47] SysML. Sysml, website. <http://www.sysml.org>, 2016.
- [48] Sebastian Theiss, Volodymyr Vasyutynskyy, and Klaus Kabitzsch. Software agents in industry: A customized framework in theory and praxis. *IEEE Transactions on Industrial Informatics*, 5(2):147–156, 2009.
- [49] Jan Thieme and Hans-Michael Hanisch. Model-based generation of modular plc code using iec61131 function blocks. In *Proceedings of the International Symposium on Industrial Electronics*, volume 1, pages 199–204, 2002.
- [50] Kleantes Thramboulidis. Iec 61499 vs. 61131: a comparison based on misperceptions. *arXiv preprint arXiv:1303.4761*, 2013.

- [51] Muhammed Ucar and Robin G Qiu. E-maintenance in support of e-automated manufacturing systems. *Journal of the Chinese institute of industrial engineers*, 22(1):1–10, 2005.
- [52] Carnegie Mellon University. Software engineering institute. <http://www.sei.cmu.edu>.
- [53] Kurapati Venkatesh, MengChu Zhou, and Reggie J Caudill. Comparing ladder logic diagrams and petri nets for sequence controller design through a discrete manufacturing system. *IEEE Transactions on Industrial Electronics*, 41(6):611–619, 1994.
- [54] M Waclawiak, M McGranaghan, and D Sabin. Substation power quality performance monitoring and the internet. In *Power Engineering Society Summer Meeting, 2001*, volume 2, pages 1110–1111. IEEE, 2001.
- [55] JF Wang, W Tse Peter, LS He, and Ricky W Yeung. Remote sensing, diagnosis and collaborative maintenance with web-enabled virtual instruments and mini-servers. *The International Journal of Advanced Manufacturing Technology*, 24(9-10):764–772, 2004.
- [56] Wanbin Wang, W Tse Peter, and Jay Lee. Remote machine maintenance system through internet and mobile communica-

- tion. *The International Journal of Advanced Manufacturing Technology*, 31(7-8):783–789, 2007.
- [57] Martin Wollschlaeger and T Bangermann. A web-based maintenance portal based on an xml content model. In *Industrial Technology, 2003 IEEE International Conference on*, volume 1, pages 405–410. IEEE, 2003.
- [58] James P Womack, Daniel T Jones, and Daniel Roos. The machine that changed the world: The story of lean production: How japan’s secret weapon in the global auto wars will revolutionize western industry. *New York: Rawson Associates.-1990*, 1991.