Diogo Alfaro Cardoso da Gama Barata

Licenciado em Ciências de
Engenharia Electrotécnica e de Computadores

Product Extension Services Deployment Platform

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador : José António Barata de Oliveira,
Professor Doutor, FCT-UNL

Júri:

Presidente:	Prof. Doutor Tiago Oliveira Machado de Figueiredo Cardoso
Arguente:	Prof. Doutor João Paulo Branquinho Pimentão
Vogal:	Prof. Doutor José António Barata de Oliveira

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2015

Product Extension Services Deployment Platform

To Mónica, my family and my true friends.

Acknowledgements

A big thanks to Professor José Barata for giving me his trust and the opportunity to work deep inside an European project, showing me the other side of the coin which teached me new things and contributed greatly for my personal growth.

A very special thanks to my university friends João Almeida and Pedro Carrasco, who went along side with me through this tremendous years, filled with late night studies, deadline projects, lectures that no one, not even the teachers, knew what was being lectured there, and also for supporting me in my crazy moments and for only getting slightly mad with my insane way of deleting almost finished work near deadlines. I couldn't forget to give this two guys an extra special thanks, to Pedro Carrasco for going with me on an Erasmus adventure and, to João Almeida for being the really good friend that is always ready to help no matter what.

From the Erasmus world I couldn't forget my good friends Matthias and JF. Both far away friends that I would never forget.

For my lab colleagues with whom I have spent most of my time during my thesis year, Paulo Rodrigues, Francisco Marques, Ricardo Mendonça, André Lourenço, Eduardo Pinto, Raul Guilherme, Ricardo Pombeiro, Giovanni di Orio and André Rocha, a big and warm thanks for all the help and friendship they gave me. From this guys a special thanks to Paulo Rodrigues who I just meet in this last year and turned out to be a great friend ready to help in my insane thoughts and with what ever comes along.

For the people who entered in my new chapter of life, the business world, I would like to thanks João Coelho (a.k.a. "Johnyyy"), Joana Pereira, Miguel Júlio, Ruben Garcia and Luís Pereira. Although there wasn't a single day they didn't make fun of me asking me if my thesis was finished, they were actually being supportive and motivating me to end it while keeping the good mood.

One of the biggest thanks I could give goes to my family, specially my brother and my mother who opened the doors so that I could have a proper education, supported it and advised me all my life.

A big thanks to my big friend Sara Mendes who was like a big sister to me supporting and advising me in some big decisions and with who I spent great times with.

Last but lot least, my ultimate thanks goes to my girlfriend Mónica who helped me every time I needed and knew when I needed my time to get things done. The person who knew when to push me and when to leave me alone. The greatest support came from her, always knowing what to say to me in both good and bad moments. Thank you Mónica for being that special person.

Abstract

Due to the progresses made in the branch of embedded technologies, manufacturers are becoming able to pack their shop floor level manufacturing resources with even more complex functionalities. This technological progression is radically changing the way production systems are designed and deployed, as well as, monitored and controlled. The dissemination of smart devices inside production processes confers new visibility on the production system while enabling for a more efficient and effective management of the operations.

By turning the current manufacturing resources functionalities into services based on a Service Oriented Architecture (SOA), in order to expose them as a service to the user, the binomial manufacturing resource/service will push the entire manufacturing enterprise visibility to another level while enabling the global optimization of the operations and processes of a production system while, at the same time, supporting its accommodation to the operational spike easily and with reduced impact on production.

The present work implements a Cloud Manufacturing infrastructure for achieving the resource/service value-added i.e. to facilitate the creation of services that are the composition of currently available atomic services. In this context, manufacturing resource virtualization (i.e. formalization of resources capabilities into services accessible inside and outside the enterprise) and semantic representation/description are the pillars for achieving resource service composition. In conclusion, the present work aims to act on the manufacturing resource layer where physical resources and shop floor capabilities are going to be provided to the user as a SaaS (Software as a Service) and/or IaaS (Infrastructure as a Service).

Keywords: Cloud Manufacturing, Service-oriented Architecture, Service Composition, Device Profile for Web Services

Resumo

Devido aos progressos realizados no ramo das tecnologias embutidas, os frabicantes estão a tornar-se capazes de equipar os seus recursos industriais ao nível da linha de montagem com funcionalidades cada vez mais complexas. Esta progressão tecnológica está a mudar radicalmente a forma como os sistemas de produção são projetados e implementados, assim como, monitorizados e controlados. A disseminação de dispositivos inteligentes dentro de processos de produção confere uma nova visibilidade sobre o sistema de produção, permitindo ao mesmo tempo uma gestão mais eficiente e eficaz das operações.

Ao transformar as actuais funcionalidades dos recursos industriais em serviços baseados numa Arquitectura orientada a Serviços, a fim de expô-los como um serviço para o utilizador, o binómio de recursos/serviços industriais irá impulsionar a visibilidade da empresa industrial para um outro nível, permitindo simultaneamente a optimização global das operações e processos dos sistemas de produção e, ao mesmo tempo, suportanto a acomodação do pico operacional de uma forma facil e com um baixo impacto na produção.

Este trabalho implementa uma infraestrutura de manufactura em *Cloud* para atingir o valor acrescentado dos recursos/serviços, isto é, para facilitar a criação de serviços que são a composição de serviços atómicos actualmente disponíveis. Neste contexto, a virtualização de recursos industriais (ou seja, a formalização das capacidades dos recursos industrias em serviços acessiveis dentro e fora da empresa) e a representação/descrição semantica são os pilares para alcançar a composição de serviços de recursos industriais. Em conclusão, este trabalho tem como objectivo atuar na camada de recursos industrias onde recursos físicos e as capacidades de linha de montagem vão ser proporcionadas ao utilizador como SaaS (Software as a Service) e/ou IaaS (Infrastructure as a Service).

Palavras-chave: Manufactura em *Cloud*, Arquitectura orientada a Serviços, Composição de serviços, Perfil de dispositivos para Serviços *Web*

Contents

1 Introduction **1**

1.1 Motivation . . . 1
1.2 Research Problem . . . 2
1.3 Approach . . . 3
1.4 Dissertation Outline . . . 4

2 State-of-the-Art Analysis **5**

2.1 The Evolution of Manufacturing Paradigms . . . 5
2.1.1 Industrial Age . . . 6
2.1.1.1 Craft Production . . . 6
2.1.1.2 Mass Production . . . 7
2.1.2 Information and Post-Information Age . . . 7
2.1.2.1 Lean Manufacturing . . . 8
2.1.2.2 Mass Customization . . . 9
2.1.2.3 Mass Customization: types of Manufacturing processes . 9
2.1.2.4 Cloud Manufacturing an enabler for Mass Personalization 13
2.2 Supporting Concepts and Technologies . . . 15
2.2.1 Service Oriented Architecture (SOA) . . . 15
2.2.1.1 SOA: Definition and Basic Concepts . . . 15
2.2.1.2 SOA in Manufacturing . . . 17
2.2.2 Cloud Computing . . . 20
2.2.3 Service Composition . . . 21
2.2.3.1 Service Orchestration . . . 21
2.2.3.2 Service Choreography . . . 23
2.2.4 SOA at Device Level: Device Profile for Web Services (DPWS) . . . 24

3 Architecture Overview **27**

3.1 Deployer . . . 28
3.2 Cloud Server Manager . . . 29

3.3 Client UI 29

4 Implementation and Validation **31**
4.1 Installation 31
4.2 MOFA France service description and overview 34
4.2.1 Workstations Description 34
4.2.2 Control Organization 34
4.2.3 Control Configuration 35
4.2.4 Description of the Operations 37
4.3 Deployer 38
4.3.1 Device Explorer 40
4.3.2 Device Virtualization 40
4.3.3 Device Repository 41
4.3.4 Device Handler 41
4.4 Cloud Server Manager 41
4.5 Client UI 42
4.6 Tests and Results 43

5 Conclusions and Future Work **47**
5.1 Conclusions 47
5.2 Future Work 48
5.3 Scientific Contributions 48

6 Appendix 1 - WSDL file for the Crane Service **57**

List of Figures

2.1 Manufacturing business paradigms until the present day adapted from [Di Orio, 2013] . . . 6
2.2 Layered framework for implementing Cloud Manufacturing [Xu, 2012] . . 14
2.3 Cloud Computing and Cloud Manufacturing in a nutshell . . . 15
2.4 Cloud Computing Service Delivery Model [Marinos and Briscoe, 2009] . . 21
2.5 Orchestration behaviour example [Cândido et al., 2009] . . . 22
2.6 Choreography behaviour example [Cândido et al., 2009] . . . 23
2.7 Device Profile for Web Services protocol stack [Jammes et al., 2005] . . . 26

3.1 Infrastructure Architecture . . . 27
3.2 Detail of the *Deployer* Architecture . . . 29
3.3 Detail of the *Cloud Server Manager* Architecture . . . 30

4.1 MOFA France educational kit - (a) MOFA France kit. (b) MOFA France components overview . . . 32
4.2 MOFA France new control equipment - Inico S1000 PLC rack . . . 33
4.3 MOFA France distributed service-oriented control architecture with respective services . . . 35
4.4 Inico S1000 programming environment - example of the configuration of a service . . . 36
4.5 *Deployer* GUI . . . 39
4.6 *Cloud Server Manager* GUI . . . 42
4.7 *Cloud Server Manager* Entity Diagram . . . 43
4.8 *Client UI* GUI . . . 44

List of Tables

2.1 Comparative Analysis between SOA and MAS [Ribeiro et al., 2008] 19

4.1 Description of the service operations 38

Listings

4.1 Inico S1000 programming environment - example of Input message in XML-based format 37
4.2 Inico S1000 programming environment - example of ST program 37
6.1 Inico S1000 programming environment - example of ST program 57

List of Notations

AWS	Amazon Web Services
BMS	Bionic Manufacturing System
$BPEL$	Business Process Execution Language
CIM	Computer Integrated Manufacturing
$DPWS$	Device Profile for Web Services
EAS	Evolvable Assembly System
$EC2$	Elastic Compute Cloud
EPS	Evolvable Production System
ERP	Enterprise Resource Planning
FAM	Flexible Assembly System
FMS	Flexible Manufacturing System
HMI	Human Machine Interface
HMS	Holonic Manufacturing System
$HTTP$	Hypertext Transfer Protocol
$IaaS$	Infrastructure as a Service
ICT	Information and Communications Technology
IMS	Intelligent Manufacturing System
IT	Information Technology
JBI	Java Business Integration

$JDBC$	Java Database Connectivity
$JINI$	Java Intelligent Network Infrastructure
$JMEDS$	Java Multi Edition DPWS Stack
JMS	Java Message Service
MAS	Multiagent Systems
MES	Manufacturing Execution Systems
$NIST$	National Institute of Standard and Technology
$OWL-S$	Ontology Web Language for Services
$PaaS$	Platform as a Service
QoS	Quality of Service
RMS	Reconfigurable Manufacturing System
RTU	Remote Terminal Unit
$SaaS$	Software as a Service
$SCADA$	Supervisory Control And Data Acquisition
SOA	Service Oriented Architecture
$SOAP$	Simple Object Access Protocol
ST	Structured Text
$UDDI$	Universal Description Services Definition Language
$UPnP$	Universal Plug and Play
$W3C$	World Wide Web Consortium
$WS-CDL$	Web Services Choreography Description Language
$WSDL$	Web Services Definition Language
$XaaS$	Everything as a Service
XML	eXtensible Markup Language

1

Introduction

1.1 Motivation

Globalization has triggered an unprecedented and unseen degree of changes in both structural and organizational layers of a manufacturing company. The market turbulence, volatility and unpredictability together with the customer demands for higher quality and highly customized products at lower costs and minimum time-to-market delay are drastically changing the way production systems are designed and deployed [Pine, 1999], i.e. manufacturing companies are rethinking their processes to enable rapid response and adaptation to their markets and costumer needs. Some of these changes pass through creating a strong cooperation among different manufacturing companies to share knowledge, expertise and resources to benefit in a collective manner [Wang, 2013, Wu et al., 2013, Jassbi et al., 2014]. Although such cooperation brings great benefits it also brings some disadvantages due to the cross-boarder collaboration that led to a geographically distributed shop-floor environments, making the manufacturing process highly decentralized and dependent of the partners involved.

As stated in [Jassbi et al., 2014], a better flow of information can lead to a better/optimized flow of materials while improving the efficiency and effectiveness of the supply chain operation. To achieve a better flow of information the manufacturing companies need to have a robust and fully available communication infrastructure that allows the integration of information between all the actors inside the manufacturing layers [Vincent Wang and Xu, 2013]. This can be achieved with the use of Cloud Manufacturing, a new manufacturing paradigm that allows the virtualization of manufacturing resources in terms of their capabilities. By virtualizing the manufacturing resources they can be shared through out the company, allowing this way a high flow of knowledge sharing about the manufacturing processes ongoing. In such a way the manufacturing

partners can plan their manufacturing processes according to the other partners processes and delays, giving to the manufacturing company a better intra-enterprise collaboration, higher flexibility and agility in the management of the manufacturing processes and transparency between all the layers of the manufacturing enterprise.

The Cloud Manufacturing paradigm can represent a solution for the demand of globalization and the decentralization of manufacturing companies by enabling a new form of interaction between enterprises improving their agility. Once Cloud Computing emerged as the latest computing paradigm providing on-demand computing services with high reliability, scalability and availability in a distributed environment, Cloud Manufacturing represents the extension of this concept applied to the manufacturing industry to achieve the demand for globalization enabling the sharing and the aggregation of geographical distributed resources over the network [Xu, 2012, Vincent Wang and Xu, 2013].

The implementation of the Cloud Manufacturing paradigm in a manufacturing company will imply a full transformation from a production-oriented manufacturing to a service-oriented manufacturing. By turning it into a service-oriented manufacturing paradigm manufacturers will be able to accomplish full-scale sharing, free transaction, and on-demand use of various manufacturing resources and capabilities [Tao et al., 2013] allowing a geographical distributed collaborative design.

The purpose of this dissertation is to present the research and development of a Product Extension Services (PES) Platform that will be developed in the framework of the ProSEco project supported by a Cloud Manufacturing approach. Therefore this dissertation has the objective of creating a infrastructure to support PES deployment and to allow/aid the creation of service composition.

ProSEco is a European FP7 Project that aims to address this needs with its objective to provide a novel methodology and a comprehensive information and communication technology (ICT) solution using a Cloud Manufacturing approach for collaborative design of product-services and their production processes as well as the effective implementation of innovative services in order to strengthen manufacturing companies' competitiveness in market sharing.

1.2 Research Problem

Nowadays manufacturing companies are becoming more aware that a paradigm shift is needed in their production systems in order to cope with the emerging requirements imposed by the globalization. This made manufacturing companies become aware that the shop-floor equipment is characterized by a high degree of diversity in device functionality, form factor, network protocols, input/output features, as well as the presence of many heterogeneous hardware and software platforms making the shop-floor customization too hard and time demanding to address the customers demands in an affordable price and suitable time. This shop-floor configuration combined with a large base of installed devices like industrial automation, origins a patchwork of technology islands

known by its poor interoperability and scalability [Candido et al., 2011].

To address this problem, in this document a Cloud Based Infrastructure is proposed to control and monitor the manufacturing resources of the shop floor system as well as the deployment of the resources virtualization in the cloud, enabling the remote access of the manufacturing resources.

This Infrastructure should provide solutions to the most common issues presented in the manufacturing companies concerning:

- Optimization of the activities and processes during the production.
- Improved data management and accommodating the operational spike easily and/or with reduced impact on production.

1.3 Approach

In this document, a cloud-based infrastructure is proposed. The infrastructure has to be able to virtualize the resources so that they can be accessed through the internet. It should also have the capability to control, monitor and manage these resources externally.

The approach followed to achieve the Infrastructure presented in this dissertation is constituted by the following steps:

1. **Components Requirements:** an investigation about the best suited architecture as well as its components requirements and functionalities was made to understand how they would act and interact with each other and the surrounding environment.
2. **State-of-the-Art Analysis:** a comprehensive analysis about existent and/or available theoretical approaches, tools and services has been conducted in order to provide the necessary background upon which the Infrastructure solution can be designed and implemented.
3. **Design and Implementation:** the components (*Deployer*, *Cloud Server Manager* and *Client UI*) are individually designed and implemented according to their individual requirements defined in the previous step.
4. **Experiments:** the infrastructure is put to test to assure that is working properly and according to the specified requirements.
5. **Validation:** finally, the outcome of the experimentations is verified to determine if the infrastructure is working as specified in the previous steps.

Although this document addresses Service Composition on the State-of-the-Art, this process wasn't developed since the main focus of the work carried wasn't to develop this process, but to develop an infrastructure that enables it. Therefore, some research needed to be made on this topic.

1.4 Dissertation Outline

This dissertation is organized as follows:

- Chapter 2: presents a review of the state-of-the-art for the technologies fundamental for supporting the design and development of the PES Deployment Platform.
- Chapter 3: gives an overview of the architecture adopted for the infrastructure.
- Chapter 4: describes the components used and the work done during the implementation and validation of the PES Deployment Platform.
- Chapter 5: gathers a set of conclusions, main contributions addressed by the dissertation and future work to be done on the subject of the PES Deployment Platform.

2

State-of-the-Art Analysis

In the old days consumers were the ones dictating the manufacturing process, demanding for products that would fulfil their needs, with each product being unique and personal, but as time passed, manufacturers started offering limited choices at lower price, reducing production time as every product was standardize. Nowadays, this mindset is changing back to its original state where consumers are once again demanding for products according to their needs and desires forcing the manufacturing industry to adapt to this changes.

2.1 The Evolution of Manufacturing Paradigms

According to [Koren, 2010], industry has always replied to market and societal changes and imperatives by developing new manufacturing processes to produce products, and new manufacturing paradigms to sell them, thus, throughout history, the manufacturing industry has undergone several revolutionary manufacturing paradigms mainly spread over three different ages: industrial age, information age, and post-information age. In Figure 2.1 is possible to see the time line of the different ages as well as the manufacturing paradigms raised to cope with the market of that time.

The start and end of each age as well as the birth and death of each manufacturing paradigm is unclear, i.e it is impossible to identify exact points in time where one age and/or manufacturing paradigm can be considered terminated and the next one starts. On the other hand, during the transitions it is normal to see an overlapping of paradigms and/or ages that can last for an uncertain amount of time due to one paradigm being progressively abandoned while the new one raises as the adopted one.

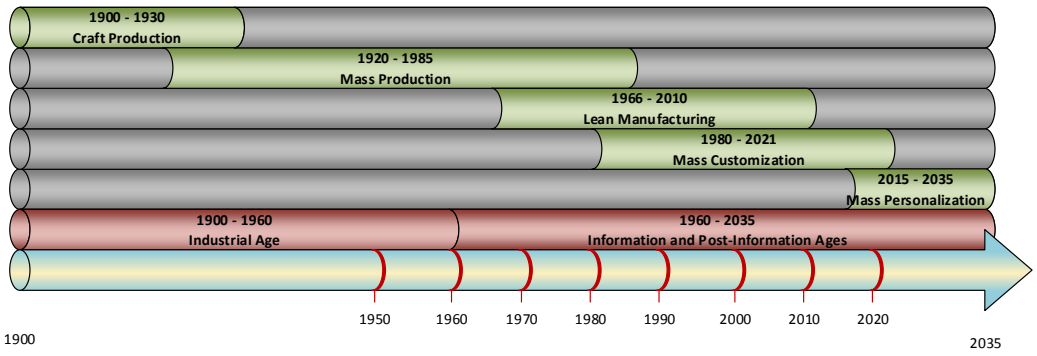


Figure 2.1: Manufacturing business paradigms until the present day adapted from [Di Orio, 2013]

2.1.1 Industrial Age

The industrial revolution, a period from late eighteenth century to early nineteenth century where major changes in technologies, transports, mining and agriculture had a profound effect on the socio-economic and cultural conditions in most of the developed countries, is considered to be the trigger responsible for the birth of the industrial age [De Masi, 2000] with its shift from manual-labour-based economy to a machine-based manufacturing, in other words, from craft production to mass production.

One invention stood out as the major breakthrough that set off the industrial revolution, the invention of the steam machine. The steam machine provided the creation of steamboats and locomotives, which made transportation easier and faster, shortening the time of travel and therefore making places closer together. By making the world smaller, the industrial revolution started to spread throughout the world allowing raw materials like iron, timber and oil to become available in bigger quantities and on demand as well as labour force to become easily available leading to the industrialization.

2.1.1.1 Craft Production

Craft production is the manufacturing paradigm of the end of the nineteenth century, mainly mastered and dominated by Europe, where highly skilled workers (Craftsmen) capable to participate in the different phases of the production process, used general-purposed and highly-flexible machines to create the exact products requested by the customer. These products were made on demand and one item at a time, consequently these products weren't standardized and therefore the production process was a "one of a kind" process where unique products were produced implying high prices with the consequence that only few customers could afford them [Di Orio, 2013].

In spite of the fact that craft production could create a great variety of products, this flexibility came with the cost of an incredible slow production rate and high prices depending on the products produced [Ribeiro and Barata, 2011]. Not only the production

rate was slow due to the production being made one item at a time, but also because of the challenge that changing or adapting the manufacturing process represented. Since every product was "one of a kind", every time a new product needed to be produced the manufacturing process needed to be changed in order to be fitted to the new product requirements. Due to this limitations, craft industries, although flexible, showed big difficulties to quickly adapt their manufacturing processes to market variations, like an increase of demand, or product changes.

2.1.1.2 Mass Production

Mass production became characterized by Henry Ford's acknowledgement and proof that producing in a continuous and synchronized assembly line, using interchangeable parts, where workers were assigned to specific and systematized tasks would increase the global quality of the product while producing faster and allowing prices to have significant drops [Ford and Crowther, 1988, Gross, 1996]. By focusing on a single product with no-diversification whatsoever, manufacturers were able to create systematized processes capable of producing large amounts of the same product in a very efficient way at a lower price, making this paradigm famous for its low prices per unit, great quantities, fast manufacturing and no customization at all from the customers side. As Henry Ford once said: "Any customer can have a car painted any colour that he wants so long as it is black.". This sentence represents a good statement for the lack of diversity of products characterized by this paradigm as well as the turning point where the customer is the one that must adapt to the product and not vice versa as seen in craft production where the product was made to suit the customer's request. Although mass production didn't had any concern about the customers needs and preferences, its philosophy to assume that there will always be a possible costumer willing to buy, lowered the prices which allowed more people to be able to afford these products, making a significant increase in sales and market power.

2.1.2 Information and Post-Information Age

With the dissemination of the means of communication (based on electronics and computers) and computers among the masses of people in the seventies of the last century, a new age is believed to begin where computer technology can be seen as the main catalyst of a social and economic movement called the information age. For some authors [Kornhauser, 1959, Bell, 1962, Hames, 1994] the mass society, a consequence of a mature mass production, was the most important cause for the transition from the industrial age to the information age. With mass society generating a progressive and general increase in customers welfare, an increasing demand for more sophisticated and customized products was verified, making product quality and customization a fundamental factor of choice for customers.

The age of uniformity and standardization gave way for an age of turbulence where

variety, customization, better quality, continuous innovation and strong competition in market sharing are the cornerstones. It is believed that the change from the industrial age to the information age has its basis set on the dissemination of electronics and computers, mainly because these became so disseminated that they improved the means of communication and therefore organization became possible. This led to a social and economic movement where people became more sophisticated, aware of their own rights and therefore more demanding.

2.1.2.1 Lean Manufacturing

Lean manufacturing was first seen in Japanese manufacturers, more specifically in manufacturers led by Toyota, where more efficient system standards were applied to eradicate all waste of work, time and material as well as to detect errors in an easier and quicker way. More relevant, comparing to mass production, the quality of the product made using lean manufacturing paradigm was superior [Womack et al., 1990].

The main concern around lean manufacturing is to avoid *Muda*, which is the Japanese word for waste, representing every activity that absorbs resources but doesn't create any value. These wastes can result from countless sources, such as: overproduction, if a factory is producing more than the market requires, which increases inventory and labour costs; waiting time, when parts are held in buffers or taking too long in a process and don't arrive in time for the next task; product defects which can result in a waste of product as well as extra costs for its elimination, sometimes a complex process involving extra activities is needed to eliminate the defected product which may lead to delays in the production of other products; inefficient processing that results from a bad distribution of tasks in each machine/worker and bad process planning; underutilized workers, people whose job aren't bringing any value to the factory or can be made by another person without affecting its work load. Therefore, if the manufacturing company removes or reduces the waste, the costs of production and time to market are reduced enabling a significant reduction in the products price.

Comparing to the mass production paradigm, the main difference is the less of everything (i.e. tool investment, manufacturing space, manufacturing time, human resources, etc.) focusing on goals such as continually declining costs, zero defects, zero inventories (items are produced only when they are needed) and endless product variety [Womack et al., 1990].

Another aspect of lean manufacturing is the way the production line (shop floor) and its workers are organized. Shop floor workers are organized into teams with a team leader rather than a foreman, as occurred in mass production. The workers are polyvalent and able to execute the various tasks assigned to the team. This generally provides a greater sense of fulfilling in the workers since they are not confined to the repetitive execution of the same task as in mass production. Furthermore, teams have the right to stop the assembly line, whenever they think it is necessary, for instance, as when they

are repairing it. Workers are also stimulated to participate with suggestions to improve the process. This continuous improvement strategy can be effective because workers, if properly motivated, can contribute substantially since they are the ones that truly master the processes being taken care of [Ribeiro and Barata, 2011]. Although lean manufacturing advertises a less monotonous working environment and conditions as an advantage it actually has a side effect on workers, since pressures on the managers to reduce wastes results in an increase of the anxiety levels towards a more efficient production.

2.1.2.2 Mass Customization

As the economy grows, society became wealthier and started to want more than just standard low cost products, demanding for a larger variety of products to choose from, in order to have their preferences fulfilled at the same low cost as in mass production. In response to the markets diversification and society changes, manufacturers transformed their standard products into new and more complex ones which included the same standard products but with a set of extra features and/or packages that, in turn, are offered to customers to configure their own products and fulfil their requirements.

Mass customization, known as the concept of providing personalized products at a reasonable price, was popularised by Joseph Pine II [Pine, 1999] for whom mass customization was a new way of doing business with its core set on a fast increase of variety and customization of products, without increasing costs. The identification and fulfilment of the individual customers requirements and desires without sacrificing efficiency, effectiveness and low costs represent the basic requirements and challenges of this paradigm.

Differently from what was seen in previous paradigms were products where one-of-a-kind (craft production) or equal for everyone (mass production and lean manufacturing), in mass customization the goal is for the development of a standard product equal for everyone but with slight changes in the final product selected by the customer, making it not fully one-of-a-kind but enough for the customer to have a product that fulfils its requirements and preferences. Therefore the major objective of mass customization is to take advantage from the mass production paradigm assets benefiting from its production low costs while adding configurations capabilities to the final assembly.

2.1.2.3 Mass Customization: types of Manufacturing processes

For many years, several manufacturing processes have been designed and implemented to satisfy mass customization requirements while improving and/or ensuring manufacturing companies competitiveness and position in the market sharing. To achieve a better response to the markets demands, manufacturing companies started to make large investments in technology, more specifically in automation and software to manage their manufacturing production processes. However, there wasn't a global strategy for the integration between the manufacturing systems components, i.e. both the software and

hardware used were heterogeneous and therefore the component integration was complex or incompatible.

To face the heterogeneity of the manufacturing system components, the Computer Integrated Manufacturing (CIM) paradigm [Browne et al., 1988, Ranky, 1990, Scheer, 1991, Mitchell, 1991, Waldner and Duffin, 1992, Camarinha Matos et al., 1995] emerged to create a global architecture capable of modelling the different tasks in a factory and provide an integrated view of the manufacturing company. The arrival of CIM became a valuable contribution to the increasing competitiveness of manufacturing companies, mainly by the introduction of automation and a wider use of computers, providing a reference architecture for design and implementation of manufacturing processes while, at the same time, introducing a certain degree of flexibility in which is considered to be the starting point for the implementation of the following manufacturing processes that appeared.

2.1.2.3.1 Flexible Manufacturing/Assembly System (FMS/FAS)

Manufacturing companies became aware of the importance that flexibility could have in their production systems to face the increasingly tendency towards diversified products, with varying demand and market unpredictability. Therefore, manufacturing companies, started to focus on the production of small volumes of a high mix of products instead of great volumes of the same product. To achieve this goal the manufacturing production systems needed to be changed in order to become more flexible, i.e. they needed to have the ability to quickly adapt the manufacturing processes to produce a range of predetermined diversified products, therefore approaches like Flexible Manufacturing System (FMS) and Flexible Assembly System (FAS) were developed.

FMS and FAS allowed manufacturing systems to handle different products as well as dissimilar demands from clients without sacrificing performance due to the design and implementation of machines capable to perform a wide range of tasks with the same performance. In detail, FMS consists in a reconfigurable set of work stations, interconnected by a transport system and an integrated computational system to control both work stations and transport systems [Upton, 1992]. Likewise, FAS is composed by assembly stations which are connected by an automated transportation system and feeding devices [Makino and Tominaga, 1995]. These processes allow for the production of several distinct products in the same system with the sacrifice of its production capabilities which are lower in comparison with a dedicated line. The accelerated development of this system was driven by the invention of programmable industry robots, new software capabilities and the widespread of computer networks which enabled the communication between all levels of the manufacturing companies, i.e. management, production, shop-floor, etc..

On the downside, it is important to take into account that these manufacturing processes are only flexible regarding the predetermined products, meaning that they are inflexible regarding the introduction of new products, due to the complexity of automatically making the required adjustments for the new process [Leitao, 2004].

2.1.2.3.2 Reconfigurable Manufacturing System (RMS)

In response to the changing manufacturing environment characterized by aggressive competition on a global scale and a rapid technology changes, the Reconfigurable Manufacturing Systems (RMS) concept was introduced [Koren et al., 1999, Mehrabi et al., 2000].

As stated in [Mehrabi et al., 2002], RMS can be defined as a "machine system which can be created by incorporating basic process modules - both hardware and software - that can be rearranged or replaced quickly and reliably." Reconfigurable systems allows for the addition, removal, or modification of specific process capabilities, controls, software, or machine structure to adjust the production capability in response to market demands or technologies. This type of systems use the concept of modular machines and open-architecture controllers which have the ability to integrate/remove new software/hardware modules without affecting the rest of the system, giving RMS the ability to be quickly converted to face new technologies, new production models, and market growth.

RMS was created with the objective of providing the needed functionalities and/or capacity when it is needed, meaning that a RMS configuration could be changed according to its needs from flexible, to dedicated, or in between. Compared to FMS, RMS stepped up by allowing the reduction of lead time for launching new systems and reconfiguring existing systems, as well as for the rapid manufacturing modification and quick integration of new technologies and/or new functions into existing systems, allowing for a rapid response to market fluctuations [Mehrabi et al., 2000].

2.1.2.3.3 Holonic Manufacturing System (HMS)

The Holonic Manufacturing System (HMS), which emerged from the framework of the Intelligent Manufacturing Systems (IMS) programme, was inspired by the work of Arthur Koestler [Koestler, 1968] where he proposed the word *'holon'* to describe the basic unit of organization in biological and social systems. This word would later on be translated by the HMS consortium into a set of appropriate concepts for manufacturing industries with the goal to provide manufacturing industries with the benefits that holonic organisations provide to living organisms and societies, i.e. stability in the face of disturbance, adaptability and flexibility in the face of change, and efficient use of available resources [Van Brussel et al., 1998].

For a better understanding of the holonic concept applied to manufacturing, the HMS consortium established a series of working definitions for the constituent entities of the holonic systems [Christensen, 1994]:

- ***Holon:*** an autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. A holon can be part of another holon and itself a combination of other holons.
- ***Autonomy:*** the capability of an entity to create and control the execution of its plans

and/or strategies.

- ***Cooperation:*** a process whereby a set of entities develops mutually acceptable plans and execute these plans.
- ***Holarchy:*** a system of holons that can cooperate to achieve a goal or objective.
- ***Holonic Manufacturing System:*** A holarchy that integrates the entire range of manufacturing activities, from order booking through design, production, and marketing to realize the agile manufacturing enterprise.

With the manufacturing systems requiring for higher adaptability and flexibility, a HMS with its goals can represent a step forward for the development of reconfigurable, scalable, flexible and responsive manufacturing systems. An extensive review about HMS can be found in [Babiceanu and Chen, 2006].

2.1.2.3.4 Bionic Manufacturing System (BMS)

Bionic Manufacturing Systems (BMS) [Ueda, 1992, Okino, 1993] was inspired by the functioning of natural organs where the structure and work of natural life exhibits an autonomous and spontaneous behaviour, as well as a social harmony within a hierarchically ordered relationship. This is based on the organs of a life-form seemingly acting on their own while coordinating their actions and maintaining harmony between them. On the other hand, these organs consist of components such as cells and support life forms which they are part of [Tharumarajah, 1996]. This approach highlights the idea of a hierarchical system where information travels both bottom up and top down along the hierarchical chain.

2.1.2.3.5 Evolvable Manufacturing System: EAS/EPS

In the more recent years, a new manufacturing paradigm emerged with evolution as its keyword, and with the objective to provide agility in both assembly and production systems. The Evolvable Assembly/Production System (EAS/EPS) paradigm has been widely studied by several authors [Onori, 2002, Barata et al., 2006, Frei et al., 2007] with the proposition of a solution based on many simple, re-configurable, task-specific elements (system modules), that allow for a continuous evolution of the assembly/production system [Onori et al., 2006].

EAS/EPS uses the aggregation of many small and simple entities (modules) to enable a given functionality, that can also quickly disappear if some of these entities are removed. Therefore functionalities are simply created by forming different formations, or coalitions of entities [Onori et al., 2006]. In other words, EAS/EPS is a system that can dynamically adapt itself to new products and production scenarios allowing the evolution of the system together with the environment, i.e. it can add and remove manufacturing modules in response to changes in production orders and plans at run-time without

the need to completely stop the system to reprogramme/reconfigure the manufacturing tasks.

This approach focus on targeting agility through modularity and stepwise evolution based on the knowledge that modularity can enhance the evolution of systems and avoid that malfunctions and/or improvements in the system won't jeopardize the proper evolution of the system as a whole as well as the system welfare [Neves and Barata, 2009].

2.1.2.4 Cloud Manufacturing an enabler for Mass Personalization

With the continuous trend for more and more customized and/or personalized products, customers are getting more involved in the manufacturing processes, being directly and actively involved in the design of the products. This will lead to a paradigm of mass personalization where manufacturing processes will be designed to be as flexible and agile as possible, therefore a decoupled production process will be needed in order to handle high volumes of product with high variety for satisfying all kinds of customers and face the markets turbulence and unpredictability [Jassbi et al., 2014]. In this new paradigm the manufacturing company wont be the one concerned about the design process of the product, on the contrary, this process will be split between the manufacturing company, which is responsible for the product architecture, basic modules and their interfaces, and the customer which, in turn, creates his own product by selecting and composing the available modules.

To cope with this upcoming paradigm, Cloud Manufacturing arises as a new networked, service-oriented, customer centric and demand driven paradigm where manufacturing resources and capabilities are virtualized as services and turned available in the cloud to users everywhere, representing a promising enabler for the mass personalization paradigm.

Cloud Manufacturing represents the extension of the concept of Cloud Computing (a further detailed analises can be found in section 2.2.2) applied to the manufacturing industry. This concept was created to achieve the demand for globalization by transforming the manufacturing business into a new paradigm where manufacturing capabilities and resources are componentized, integrated and optimized globally, making it accessible to users everywhere [Vincent Wang and Xu, 2013]. As argued in [Tao et al., 2011], there is a big difference between Cloud Computing and Cloud Manufacturing concepts. In Cloud Computing the resources are primarily computational resources (e.g. servers, storage, network, software, etc.) that are provided in the form of services belonging to one of the three different categories, namely: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (Saas). The resources of a Cloud Manufacturing system are manufacturing resources, i.e. physical manufacturing devices, machines and more in general systems abstracted in terms of their functionalities and capabilities that are provided to the user in the form of a Cloud Manufacturing service belonging to IaaS, PaaS, and Saas. A layered framework for implementing Cloud Manufacturing consisting

of four layers can be considered [Xu, 2012](see Figure 2.2).

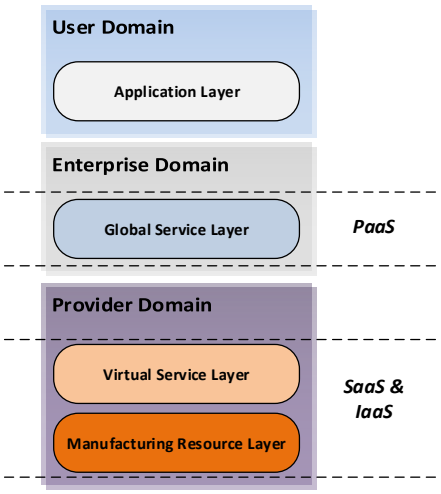


Figure 2.2: Layered framework for implementing Cloud Manufacturing [Xu, 2012]

The Manufacturing Resource layer contains the physical manufacturing resources and the shop floor capabilities that are provided to the user as SaaS and/or IaaS. The virtual service layer is responsible for virtualizing the manufacturing resources and encapsulate them into cloud manufacturing services that in turn are provided to the Global Service Layer. Therefore, the Global Service Layer is responsible to manage the Cloud Manufacturing services. Finally the Application Layer is the entry point of the manufacturing companies and provides to the user the possibility to build/construct manufacturing applications from the virtualized resources.

As exposed in [Tao et al., 2013], all the Cloud Manufacturing services are aimed to be provided to the user for encompassing the whole lifecycle of manufacturing. In such a way, typical Cloud Manufacturing services can include: Design as a Service, Manufacturing as a Service, Experimentation as a Service, Simulation as a Service, Management as a Service, Maintain as a Service, and Integration as a Service (see Figure 2.3).

Thereby, the Cloud Manufacturing paradigm and concept provides a collaborative network environment (the Cloud) where users can select the suitable manufacturing services from the Cloud and dynamically assemble them into a virtual manufacturing solution to execute a selected manufacturing task. In this scenario, the Cloud Manufacturing paradigm provides a new business model moving from the traditional product-oriented manufacturing to a service-oriented manufacturing [Xu, 2012] where the key characteristics are: better intra-enterprise collaboration, higher flexibility and agility in the management of the enterprise operations and supply chains, and transparency between all the layers of the manufacturing enterprise. Hence, Cloud Manufacturing can be defined as: "a model for enabling ubiquitous, convenient, on-demand network access to a shared

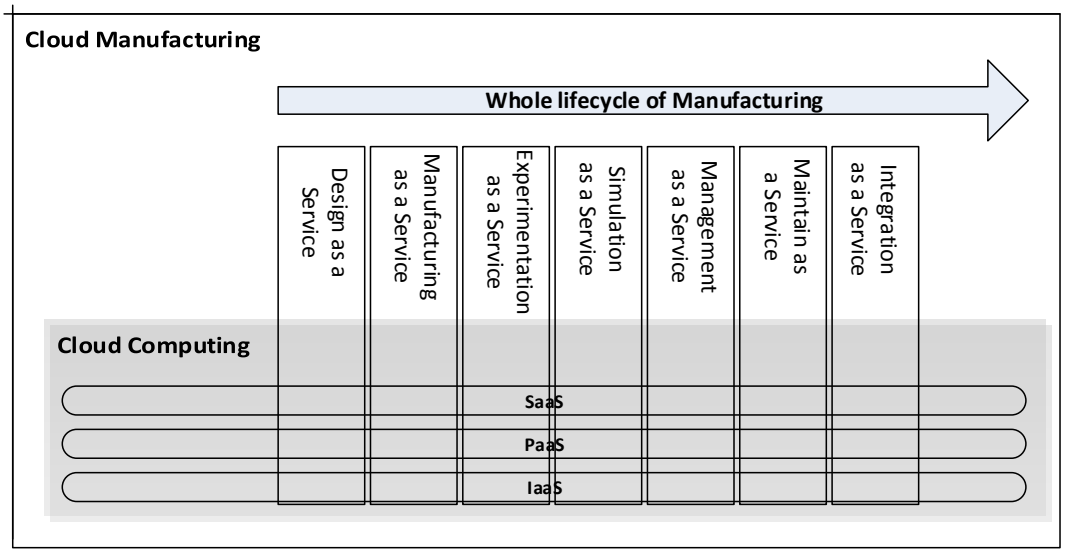


Figure 2.3: Cloud Computing and Cloud Manufacturing in a nutshell

pool of configurable manufacturing resources (e.g. manufacturing software tools, manufacturing equipment and manufacturing capabilities) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [Xu, 2012].

What is important to understand is that Cloud Manufacturing is much more than just store and retrieving data using services, it is the deployment of manufacturing resources in the form of services that will enable the design, simulation and production of custom made products designed by the customers. Therefore Cloud Manufacturing is driving us towards a mass personalization era where customers will have access to this manufacturing services and design their desired products that fulfil their own requirements.

2.2 Supporting Concepts and Technologies

2.2.1 Service Oriented Architecture (SOA)

2.2.1.1 SOA: Definition and Basic Concepts

Service Oriented Architecture (SOA) [Erl, 2005, Papazoglou and Heuvel, 2007, Josuttis, 2007] represents an emerging approach that addresses the requirements for loosely coupled, standard-based, and protocol-independent distributed computing with its promising architectural designs for rapid integration of data and business processes.

SOAs are being promoted as the next evolutionary step to help organizations to meet more complex challenges imposed by globalization and market fragmentation by establishing an architectural model that aims to enhance efficiency, agility, and productivity of

an enterprise by positioning services as the building block. It provides a framework/platform for accomplishing rapid system development, easy system modifications, while enhancing systems integration capabilities and overall system quality. Using the definition given by [Komoda, 2006], a SOA is a design framework for the construction of systems by combining services and using deeply ICT infrastructure as communication backbone. The principle behind SOA is for the development and implementation of a platform consisting of independent services, representing well-defined and self-contained modules providing standard operations that can be invoked by internal or external components (consumer of the services) using standard interfaces where services can be combined and recombined into different solutions and scenarios according to the needs. This solution is possible since the services don't depend from the state and/or the context of the other services [Di Orio, 2013].

The implementation and development of SOAs has been both enabled and stimulated by the existence of *Web Services* technology. As stated in [Natis, 2003], although *Web Services* do not necessarily translate to SOAs, and not all SOAs are based on *Web Services*, the relationship between the two technologies is important and they are mutually influential. In [Consortium and others, 2004], the World Wide Web Consortium (W3C) defined *Web Services* as: "a software system design to support interoperable device-to-device, device-to-system and system-to-system, e.g., machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the *Web Service* in a manner prescribed by its description using SOAP messages typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards". It is important to enhance the distinction between *Web Services* and Services, as stated in [Barry, 2003], the term *Web Services* refers to a collection of technologies such as eXtensible Markup Language (*XML*) [Bray et al., 1998], Simple Object Acess Protocol (*SOAP*) [Box et al., 2000], Web Services Definition Language (*WSDL*) [Christensen et al., 2001] and Universal Description Services Definition Language (*UDDI*) [Bellwood et al., 2002], which provides a standard mean of communication between different software applications, running on a variety of platforms and/or frameworks without being dependent of a particular hardware and/or technology, as for Services, are the endpoint of this communications, i.e. what is connected using *Web Services*.

Supported by a matured and universally accepted set of interoperability standards (e.g. HTTP, JSON, XML, SOAP, UDDI, WSDL, WS-* standards) for building, describing, cataloguing and managing reusable services, SOAs is the foundational architecture for today's mash-ups, software as a service and service-cloud [Karnouskos et al., 2012]. These standards give SOAs a distinct advantage over other architectural styles, since it makes both interoperability and scalability its intrinsic characteristics, which eases the integration of heterogeneous systems including legacy devices and systems providing a major enhancement in business agility and enables the capability to add and remove services without affecting the entire infrastructure.

2.2.1.2 SOA in Manufacturing

As stated in previous sections, nowadays manufacturing companies are acting and competing in a new challenging environment characterized by frequent changes in market demands, reduced time-to-market, increasing consumer demands for high quality and customized products at low cost, and efficient energy management. However, as stated in [Jammes and Smit, 2005], this trend imposes the demand for an increasingly more efficient, effective and particularly flexible production systems in order to efficiently face the turbulent market needs while maintaining the low cost base of heavily automated mass production techniques [Hajarnavis and Young, 2008]. As a result, the key to competitiveness is represented by the reduction of the production costs during the production systems lifecycle as well as the capability to have systems that are able to quickly respond to markets variations and demands for new customized products.

Presently, manufacturing companies are completely dependent on their Information Technology (IT) backbone. As exposed in [Krafzig et al., 2005], IT has played a major role inside any modern enterprise. Since its introduction, IT is running most of the processes of the enterprise itself, whether they're related to manufacturing, distribution, logistic, sales, customers management, accounting, or any other type of business process. However, the manufacturing enterprises ICT infrastructure is highly heterogeneous and distributed, making the communications between the different enterprise levels a complex task. This heterogeneity triggered the vertical adoption of the SOA paradigm as a desirable requirement to lead to a homogeneous communication infrastructure based on a single communication paradigm facilitating the data integration between the enterprise levels and providing a new degree of agility [Jammes and Smit, 2005].

As stated in [Cândido et al., 2009], the agile performance of an enterprise is strictly limited by the agility of its least agile building block, meaning that to be agile, all the enterprise IT levels, from business to device level, need to be agile. Thereby, the vertical application of the SOA paradigm in the context of machine to machine communication at shop floor level, and information integration enabler between this level and the higher levels (Resource Planning level and Business level) of a manufacturing company is fundamental. The connection between shop-floor devices and the enterprises services is essential to create more sophisticated high level services and to support more reliable decision making and closed loop control as well as process management, guaranteeing an easier and faster responsiveness and reactiveness of the whole enterprise, allowing to better face the challenges imposed by a competitive environment dominated by change and uncertainty as defined in [Goldman et al., 1995].

As listed in [Jammes and Smit, 2005], the most important requirements to be tackled by the manufacturing plants of the future include:

- Inter-enterprise dynamic integration capabilities;
- Cross-enterprise collaboration;

- Support of heterogeneous yet interoperable hardware and software environment;
- Business Agility through a production system flexibility, adaptability and reconfigurability;
- Scalability by adding or reducing resources without disrupting operations;
- Fault tolerant and efficient and effective recovery from failures in real-time production conditions;

Based on the fundamental set of requirements addressed above, nowadays manufacturing companies are engaged in an innovation race to implement more and more exclusive and efficient production systems. As stated in [Di Orio, 2013], several manufacturing processes, based on the most diverse technologies, architectures, approaches and methodologies, have been designed and implemented through the years by researches and practitioners to satisfy mass customization requirements. Some of this manufacturing processes are focused on improving responsiveness, reconfigurability and lead time, while others are focused on improving the final product quality, optimization of production activities, waste elimination, integration of secondary processes in the main control and, improving the visibility inside the manufacturing companies facilitating the information flow between all the layers of a manufacturing company. From the paradigms thought to improve flexibility and agility inside the manufacturing enterprises, Multiagent Systems (MAS) and SOAs have been considered by [Ribeiro et al., 2008] as the most promising approaches to satisfy these paradigms, even though each one has its own advantages and disadvantages according to the application scenario. A comparison between MAS and SOA made by [Ribeiro et al., 2008] can be seen in table 2.1.

Combining the results from table 2.1 and the needs of the ProSEco project as well as the task in hands for this dissertation, SOA emerges as the best solution due to its properties like communication infrastructure supported by Web related technologies, assured interoperability by the use of general purpose web technologies, and low computational requirements.

Although SOAs can represent a major breakthrough in manufacturing companies, it still has gaps that need to be fulfilled. As pointed out by [Ribeiro et al., 2011] there is no standard way of describing the services hosted by specific devices, i.e. although most of the description languages are generic they still imply the use of ontologies to store the system specific vocabulary which means that the information is accessible but there is not a standard way of making any semantic sense out of it. This lack of standards has a great impact on the information exchanged not only between the different manufacturing levels but also in the interaction patterns between the shop-floor components

Table 2.1: Comparative Analysis between SOA and MAS [Ribeiro et al., 2008]

Characteristics	**SOA**	**MAS**
Basic Unit	Service	Agent
Autonomy	Both entities denote autonomy as the functionality provided is self-contained.	
Behaviour Description	In SOA the focus is on detailing the public interface rather than describing execution details.	There are well established methods to describe the behaviour of an agent.
Social ability	Social ability is not defined for SOA nevertheless the use of a service implies the acceptance of the rules defined in the interface description.	The agent denote social ability regulated by internal or environmental rules.
Complexity encapsulation	Again, the self-contained nature of the functionalities provided allows hiding the details. In SOA this encapsulation is explicit.	
Communication infrastructure	SOA are supported by Web related technologies and can seamlessly run on the internet.	Most implementations are optimized for LAN use.
Support for dynamically reconfigurable run-time architectures	Reconfiguration often requires reprogramming.	The adaptable nature of agents makes them reactive to changes in the environment.
Interoperability	Assured by the use of general purpose web technologies.	Heavily dependent on compliance with FIPA-like standards.
Computational requirements	Lightweight implementations like the DPWS guarantee high performance without interoperability constraints.	Most implementations have heavy computational requirements.

which makes it hard to control without a standardized reference model. Research initiatives like SIRENA [1], ITEA SODA [2], IST SOCRADES [3], AESOP [4] as well as several authors [Colombo and Karnouskos, 2009, Karnouskos et al., 2010, Candido et al., 2011] have tackled these challenges. Most of the research made in this area has been directed to two major domains: (i) e-business and inter-enterprise interactions to improve enterprise agility and, (ii) flexible and reconfigurable automation systems based on the application of SOAs to the control, monitoring and management of manufacturing production processes.

[1] see http://www.sirena-itea.org/
[2] see http://www.soda-itea.org/
[3] see http://www.socrades.eu/
[4] see http://www.imc-aesop.eu/

2.2.2 Cloud Computing

Cloud Computing emerges as the latest computing paradigm that promises flexible IT architectures, configurable software services, and QoS (Quality of Service) guaranteed service environments. As stated in [Xu, 2012], the main goal of Cloud Computing is to provide on-demand computing services with the following main characteristics: high reliability, scalability and availability in a distributed environment. Although the term Cloud Computing was only coined in 2007, the concept is quite old and has rooted in 1960s, relating with the delivering of computing resources over the global network [Licklider, 1963]. A more formal definition of this concept and/or paradigm was given by the National Institute of Standard and Technology (NIST), where Cloud Computing was defined as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [Mell and Grance, 2009]. Taking into account this definition, it is possible to state that in Cloud Computing everything is treated as a service (XaaS). As exposed in [Armbrust et al., 2010], Cloud Computing refers to both the application delivered as a service of the internet as well as the hardware and system software that provides those services. The services in Cloud Computing paradigm can be divided in three distinct categories, according to the abstraction level of the capability provided and the service model of the providers, namely: IaaS, PaaS, and SaaS [Chou, 2010]. The service delivery model of a typical cloud based system is shown in Figure 2.4.

According to [Chou, 2010], the SaaS provides services and/or applications that can be accessed by users through Web portals, meaning that service consumers are shifting from locally installed computer programs to online (cloud) services that offers the same functionalities. The PaaS provides developers with a platform for allowing the creation, deployment and hosting of web applications. Finally, the IaaS provides the fundamental infrastructure i.e. the necessary hardware on-demand in the sense that customers can pay for using cloud resources.

Nowadays, Cloud Computing is emerging as a new paradigm for providing computing services on-demand, anywhere and everywhere enabling the sharing and the aggregation of geographically distributed resources over the network [Liu et al., 2011]. This paradigm has evolved from a relatively simple infrastructure that delivers storage capabilities to one that is economy based and aimed to deliver more complex services that rely on abstract resources. The Cloud Computing paradigm is completely changing the way enterprises interact with each other and, above all, with their customers. Therefore, it is creating new solutions and opportunities to the modern enterprises, including the manufacturing industry [Wang and Xu, 2013].

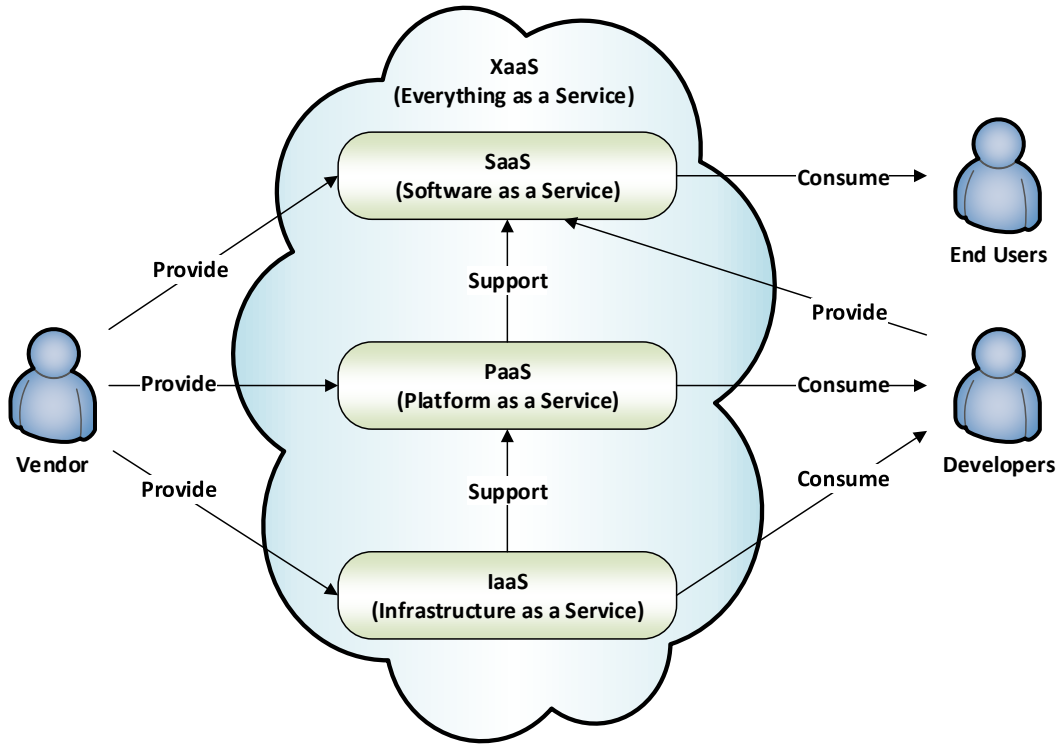


Figure 2.4: Cloud Computing Service Delivery Model [Marinos and Briscoe, 2009]

2.2.3 Service Composition

Services are the building block of a SOA, as they provide simple interactions between client and server provider. However, sometimes atomic services need to be straightforwardly combined and/or assembled in order to generate more complex ones raising the service abstraction as referred by [Cândido, 2013]. In this scenario as argued in [Dustdar and Schreiner, 2005], the term service composition is referred to the process of developing a composite service. Moreover, a composite service can be defined as the service that is obtained by the composition of the functionalities of several simplest services.

Currently in the domain of SOA-based systems, two main approaches can be used for the service composition, namely [Peltz, 2003]: orchestration and choreography.

2.2.3.1 Service Orchestration

In the orchestration approach, a central node controls a workflow that interacts with a set of services following a predetermined logic during the execution of the according complex process. The workflow logic involved in orchestration consists in several rules, conditions and events, i.e. it specifies how different entities should interoperate with the central node in order to carry out a predefined task. Therefore, from the point of view of an orchestration, the central node is the coordinator of the entire composition while the services of the composition are merely components agnostic to the fact that they are

taking part in a larger process [Bucchiarone et al., 2007]. Thereby, the process logic is centralized yet still extendable and composable, being at the same time a way to abstract a process in a single service.

A heterarchical approach is also possible by having several orchestrators at different levels of the composition, i.e. one of the partners of the orchestration can itself be another orchestrator that encapsulates and orchestrates other partners in a hierarchical way. Since each orchestrator has its own process logic, it is possible to imagine an orchestrator that sometimes during its process execution needs to invoke a service provided by another orchestrator (at the same composition level or any other), and vice-versa. An example of this approach can be seen in Figure 2.5.

Some of the advantages of using orchestration are:

- The process workflow logic is encapsulated at a single point, which makes it easy to modify without impacting the process partners;
- Orchestration can be applied recursively ("Russian dolls" composition paradigm), i.e. more orchestrators and/or services can be added to the composition;
- Preserves the autonomy of each of the process partners, that should not even be aware of each others existence, facilitating the maintenance, removal, or addition of process partners.

Several composition mechanisms, such as Ontology Web Language for Services (OWL-S), Business Process Execution Language (BPEL), and XLANG [Bronsted et al., 2010], are available for services in general and Web Services in particular.

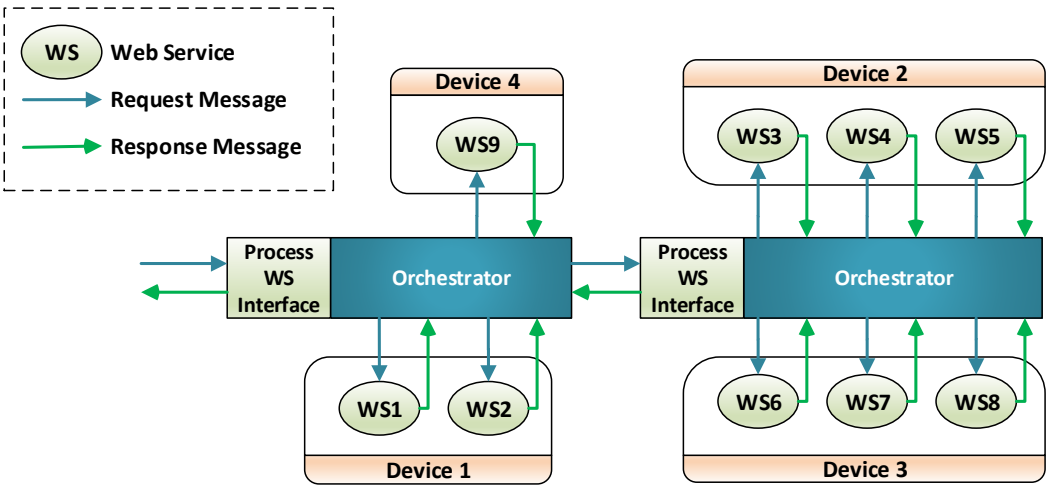


Figure 2.5: Orchestration behaviour example [Cândido et al., 2009]

2.2.3.2 Service Choreography

On the contrary of the orchestration approach, the choreography approach does not assume a central coordinator but rather defines complex tasks via the definition of the conversation that should be undertaken by each participant.

Choreography defines the same level of collaboration behaviour between distributed partners. The goal is to set up an organized collaboration between different distributed services without any other entity controlling the collaboration logic, as discussed by [Peltz, 2003]. In choreography, the aim is to expose the entire flow of interactions to all the parties involved in the composition [Bucchiarone et al., 2007]. Therefore, a choreography schema assumes that there is no owner of the global collaboration logic, consequently the composition is spread among peers, contrary to orchestration where the process execution is controlled and incorporated in a single element in a centralized manner. Also, where in orchestration the partners don't know about the existence of other partners, since they only communicate with the central coordinator, in choreography every partner knows about the existence of the other partners and communicates directly between them. A example of this approach can be seen in Figure 2.6.

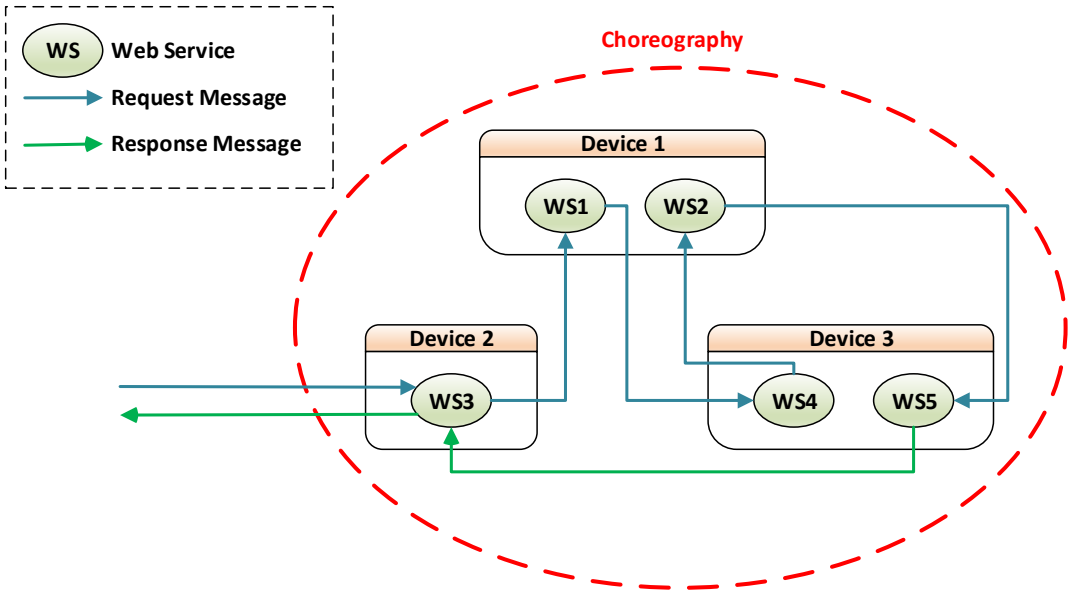


Figure 2.6: Choreography behaviour example [Cândido et al., 2009]

In order to expose a certain choreography, each service must know its own role within the current process, i.e. what the service supports and how to react or proactively execute in a particular context. These services are also referred as participants. Each possible contact between two roles in choreography is identified as a relationship. Multiple participants can assume different roles and have different relationships. The message exchange pattern of a relationship between two participants are expressed by channels containing

the necessary information so that services can determine how they can interact with other services. This channel information can be passed to other participants so that they can also join the collaboration enabling the extensibility of the composition.

The Web Services Choreography Description Language (WS-CDL) is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global point of view, their common and complementary observable behaviour, where the ordered message exchanges results in a accomplishment of a common business goal [Kavantzas et al., 2005]. In other words, WS-CDL can be used to specify the peer-to-peer collaboration of all the participants engaged in the choreography.

2.2.4 SOA at Device Level: Device Profile for Web Services (DPWS)

The Device Profile for Web Services (DPWS) was firstly submitted in 2004 as a proposal for applying Web Service standards for device networking. Since 2009, it became a standard by the OASIS Web Services Discovery [5] and Web Services Devices Technical Committee [6]. Before this, several SOA targeting device-level have been adopted such as Java Intelligent Network Infrastructure [7] (JINI) and Universal Plug and Play [8] (UPnP).

UPnP uses a series of technologies such as IP, TCP, UDP, HTTP, and SOAP to enable communication between devices. However, although UPnP could be described as a truly platform-agnostic, it uses specific protocols for both device discovery and eventing, as well as, a specific XML language for service and device description. On the other hand, there is JINI, which provides mechanisms for the discovery of services, but since it is a JAVA based solution it lacks in platform-neutrality for devices since it forces the existence of a Java Virtual Machine in every device. DPWS appeared as a new promising approach that contains equal advantages as UPnP, while adding full integration with Web Services technology which implied high acceptance among developers and platforms. Due to the integration with Web Services, DPWS is also programming language independent [Zeeb et al., 2007].

DPWS consists in a plug-and-play protocol middleware built on top of a set of Web Services standards that tackles discovery, description and control of both services and devices on local networks. DPWS defines devices and hosted services as two of its main elements where the device plays an important part in the discovery process as well as in the metadata exchange protocols, while its hosted services provide the main functionalities of the device itself and depend on their host to be discovered. To support this functionalities DPWS relies on the core Web Services standards as follows [Jammes et al., 2005]:

- *WSDL* - abstract description of services interfaces and their bindings to transport protocols;

[5] see http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01
[6] see https://www.oasis-open.org/committees/ws-dd/
[7] see http://www.jini.org
[8] see http://www.upnp.org

- *XML Schema* - defines the data formats used for constructing the messages addressed to and received from services;
- *SOAP* - protocol for transporting service-related messages formatted in accordance with the corresponding WSDL definition;
- *WS-Addressing* - concentrates the message addressing information into the header of the SOAP message envelop, allowing for the message content to be carried over any transport protocol (HTTP, TCP, UDP, ...);
- *WS-Policy* - used to express the policies associated to a Web Service in the form of "policy assertions", complementing the WSDL description of the service;
- *WS-MetadataExchange* - allows for the dynamical retrieval of metadata associated to a Web Service (i.e. description, schema and policy), providing a Web Service introspection mechanism;
- *WS-Security* - optional set of mechanisms for ensuring end-to-end message integrity, confidentiality and authentication;

To the above Web Services core protocols, DPWS adds two more Web Services protocols for discovery and eventing, namely:

- *WS-Discovery* - protocol for plug-and-play, ad-hoc discovery of network-connected resources, that defines a multicast protocol to both search for and locate devices as well as their services;
- *WS-Eventing* - defines a publish-subscribe event handling protocol allowing for one Web Service to subscribe with another Web Service to receive event notification messages;

Therefore one can say that the DPWS protocol stack is an extension of the Web Services core standards targeting specifically the device space. In Figure 2.7 is possible to see an overview of the DPWS protocol stack.

As explained in [Candido et al., 2010], the application of Web Services at device level implies significant improvements in both operational and development aspects while unifying the ICT protocols of a manufacturing enterprise by providing a single stack to communicate from device-level to Manufacturing Execution Systems (MES) or Enterprise Resource Planning (ERP) level over Web Service technology.

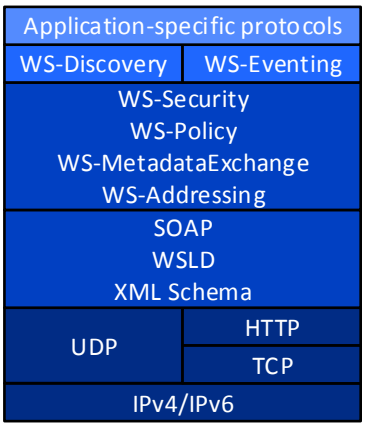


Figure 2.7: Device Profile for Web Services protocol stack [Jammes et al., 2005]

3

Architecture Overview

The following architecture was envisioned as a gateway for clients to interact with service oriented machinery over the cloud whenever and wherever they need. Therefore, the proposed architecture was thought around factories, containing service oriented machinery, deploying its services in a cloud server where users can, not only consume the services but also combine them with other services in order to create a service composition, that will allow the creation of new services according to the users needs. This thought originated the proposed infrastructure shown in Figure 3.1 which is composed by three main components/resources, namely: the *Deployer*, the *Cloud Server Manager* and the *Client UI*.

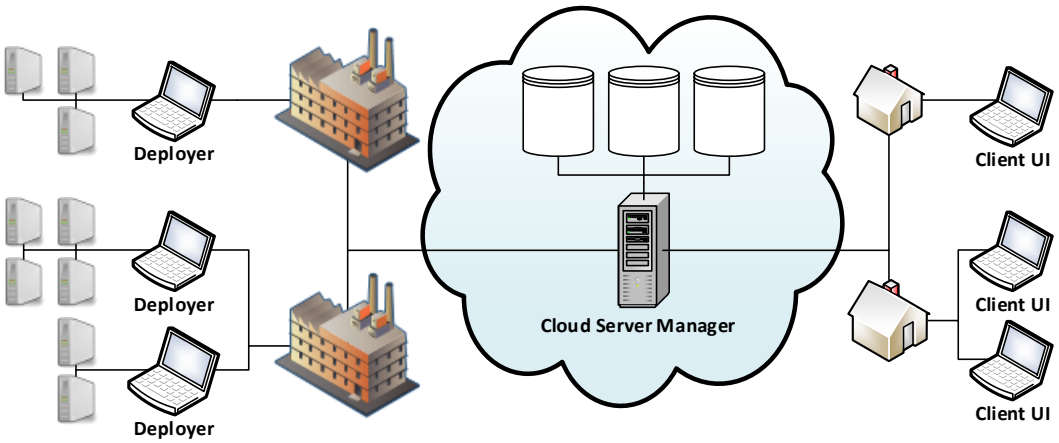


Figure 3.1: Infrastructure Architecture

These three components/resources are together responsible for:

- virtualizing the manufacturing resources available in the physical system in terms of capabilities and functionalities and deploy this virtualized resources in the cloud as cloud entities;

- exposing the capabilities of the cloud entities in order to enable their invocation by external users that are accessing the cloud so that the users can execute the desired tasks on the physical system.

The following sections will present in more detail the different components/resources of the purposed architecture.

3.1 Deployer

The *Deployer* is the component/resource responsible to scan the network (i.e the local network in which it is connected) searching for *DPWS*-enabled devices. Whenever a new device is encountered, the following tasks are executed:

- ***Virtualization:*** an abstract description will be associated to each device in order to transform the physical device into a cloud entity.

- ***Deployment:*** make the virtualized devices available on the cloud, allowing users to invoke the functionalities/services hosted by the device according to their needs.

Taking into account these tasks, the architecture of Figure 3.2 has been designed and implemented for the *Deployer*. The proposed architecture is composed by the following core task-oriented components:

- ***Device Explorer:*** it allows to search the network for available *DPWS*-enabled devices.

- ***Device Virtualization:*** once a device is discovered, this component extracts all the information from the device with the objective of creating a virtual entity with all the capabilities of the physical device.

- ***Device Repository:*** it is responsible to store all the information about the devices found in the *Deployers* network. This information will be internally used by the *Device Handler* (whenever a new request comes from the cloud) and by the *Cloud Server Manager* component/resource.

- ***Device Handler:*** it is responsible to handle requests asking for the execution of functionalities. This requests can both come from the *Cloud Server Manager* component/resource or from the *Deployer* itself by the means of an operator.

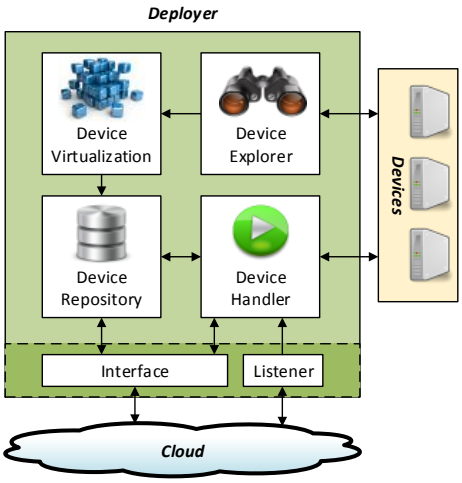


Figure 3.2: Detail of the *Deployer* Architecture

3.2 Cloud Server Manager

The *Cloud Server Manager* component/resource provides two Web Service endpoints to allow, on one side, the communication with the *Deployers* and, on the other side, the communication with the *Client UIs*. Moreover, the *Cloud Server Manager* includes a local database that is used to store all the information about the devices connected to the *Deployers* and their related functionalities. Furthermore, the database also stores all the requests performed by the users using the *Client UI* as well as all the responses to these requests performed by the *Deployers*. Thereby, the *Cloud Server Manager* is also responsible to guarantee the communication between *Deployers* and *Client UIs* acting as a Service Broker between them.

In Figure 3.3 it is possible to see the composition of the *Cloud Server Manager* component/resource. This component is composed by three modules and a database. Two of the modules are Web Services (*Deployer Services* and *Client Services*) responsible for the communication with the *Deployers* and the *Client UIs*, the third module is a Database Manager which, as the name suggests, is responsible for managing the database.

3.3 Client UI

The *Client UI* is a simple interface that connects to the *Cloud Server Manager* and allows the users to consult the deployed devices in the *Cloud Server Manager*, as well as to operate them like if they were next to the device itself playing the role of the device operator. In other words, this component tries to mimic the *Device Handler* and *Device Repository* modules of the *Deployer* component/resource in the user point of view.

For the time being, the focus of the present work was to provide an infrastructure

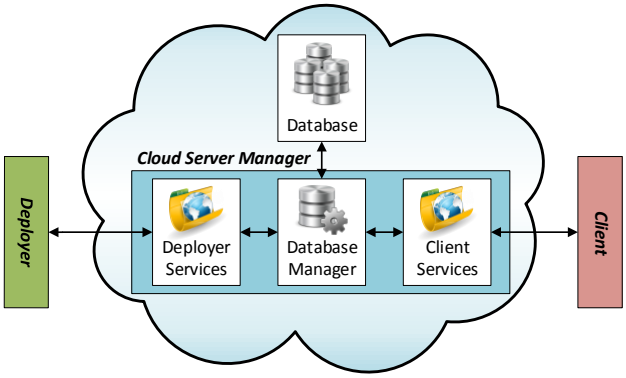


Figure 3.3: Detail of the *Cloud Server Manager* Architecture

capable of transform manufacturing resources into virtualized cloud entities capable of being deployed into the cloud as services and to expose this resource capabilities in order to enable external users to interact with them. Therefore, although the infrastructure was envisioned to enable the composition of services according to the users' needs, this module wasn't developed due to time constrains. Even though this module wasn't developed in the present work, some applications were made by [Di Orio et al., 2015] using the presented infrastructure to allow the user to create a composition of services and execute it on the physical process. In this application the *Client UI* was upgraded by adding a module composed by a graphical interface to enable users to graphically select the capabilities/functionalities they need and combine them to create a composition of services so that more complex tasks can be achieved with the use of the regular physical processes provided.

4

Implementation and Validation

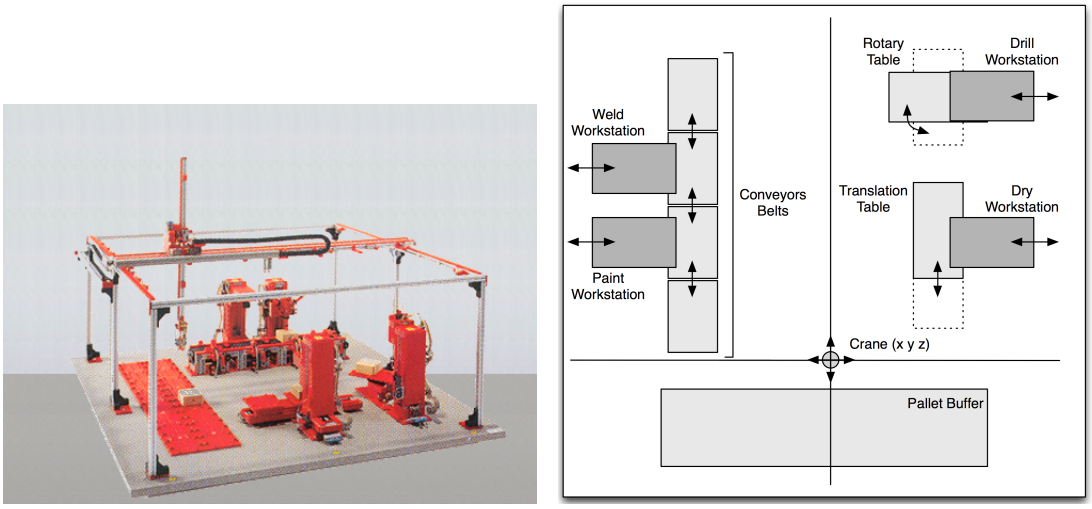
4.1 Installation

This installation comes from previous works carried by [Cândido, 2013] and [Sousa, 2014], therefore their implementation was kept mostly intact, with small modifications in the needed components.

The MOFA France kit of Staudinger GmbH was used as a proof of concept of the architecture presented in chapter 3. This educational kit simulates a flexible manufacturing system as a multi-path closed loop manufacturing circuit, capable of achieving various possible situations based on generic manufacturing tasks (see Figure 4.1(a)). Due to this flexibility, this educational kit was used to simulate a Cloud Manufacturing environment where manufacturing tasks are deployed as services and this services can be used to create different service compositions.

Staudinger models follow the modular Fischertechnik-based concept to replicate complex industrial projects in close-to-reality details. This allows systems integrators to easily discover potential problems during planning and programming, while testing suitable alternatives that can be verified quickly in a controlled environment. As verified by [Barata et al., 2008], this educational platform proved to be extremely useful for testing purposes since past experiences showed that a lot of effort can be saved if an educational platform is used. Still, key aspects of a real industrial environment should be taken into account, such as real-time and reliability, resources concurrency, mechanical and electrical relations, etc. This type of platform also has a singular advantage, because it greatly facilitates the addition and removal of any physical modules, which is much harder with a real industrial systems due to obvious logistical reasons.

This educational kit is composed by four machines that may be adaptable to different types of manufacturing tasks, a buffer area, a crane robot, local transporters (conveyors



(a) (b)

Figure 4.1: MOFA France educational kit - (a) MOFA France kit. (b) MOFA France components overview

or tables) and sensors to detect pallet positions. For this particular case study, the tasks that can be performed in these machines are *Weld*, *Paint*, *Dry* and *Drill* as presented in Figure 4.1(b). The pallets are represented by wood blocks with a carved metal ring to activate the positioning sensors as well as to be transported by the crane with the use of a electro-magnet at the tip of the crane. They represent pallets with product parts, subassemblies, or even raw materials that need to be transformed or processed. These pallets can be stored in the buffer area or be transported by the crane to the available loading positions to be processed.

For the purpose of this case study, the original control equipment, composed by a legacy data acquisition board installed in a PC that allowed the access to the kit I/O using C++ API, was replaced by a distributed service-oriented PLC solution composed by a distributed collection of Inico S1000 modules as represented in Figure 4.2. The Inico S1000 is a programmable Remote Terminal Unit (Smart RTU) device which offers process control capabilities, as well as a Web-based Human-Machine Interface (HMI), support for Web Services, real-time control and field data processing and web-based monitoring. The Inico S1000 hardware configuration includes a 32-bit CPU running at 55 MHz and 8 MB of available flash memory, 10/100 Mbit Ethernet port, 8 digital inputs and 8 digital outputs.

Besides handling typical I/O processing, the Inico S1000 also supports XML/SOAP interface based on DPWS to ease up the integration of industrial processes in a SOA context. However, this modules have some hardware limitations, namely: they only support two threads for input messages, therefore they can only handle up to two messages simultaneously; and there is only one shared thread for both outgoing events and output messages, which means that if one output message is waiting for a response, the other

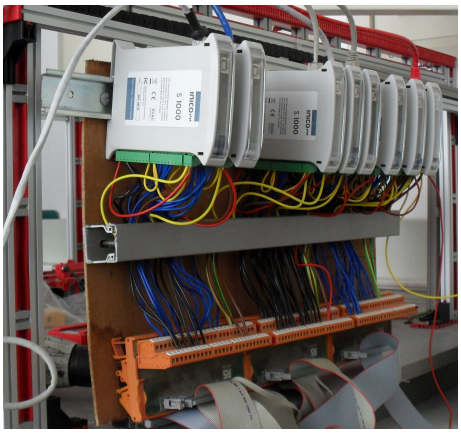


Figure 4.2: MOFA France new control equipment - Inico S1000 PLC rack

output messages and events are waiting in a queue. The control programs can be defined using the integrated browser-based editor supporting IEC61131-3 Structured Text language and configured to be triggered whenever a linked service operation is invoked. This equipment also supports the triggering of events within control programs.

For the PC-side implementations the JMEDS (Java Multi Edition DPWS Stack) framework from [WS4D, 2012] and Apache CXF services framework from [Apache, 2009] were used. JMEDS is a framework that allows the implementation and running of web services based on DPWS specification, such as, Services, Devices, and Clients. JMEDS supports all Java platforms including Java ME (Java Micro Edition) CLDC (Connected Limited Device Configuration). For the purpose of this case study, JMEDS enables the interaction with the Inico devices. Apache CXF is an open source services framework that helps to build and develop services using frontend programming APIs, like JAX-WS and JAX-RS, which are able to speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS (Java Message Service) or JBI (Java Business Integration). In this case study, Apache CXF is used to enable the communication over the network between the different components (*Deployer*, *Cloud Server Manager* and *Client UI*).

For the Network-side a commercial cloud, the Amazon Web Services (AWS), was used. The AWS is a cloud computing platform composed by a collection of computing services. For this scope the Amazon Elastic Compute Cloud (EC2) service was used. Amazon EC2 is a service that allows clients to use virtual computers allocated in Amazon's servers to run their own computer applications. In this case the *Cloud Server Manager* presented in the architecture purposed in chapter 3 is allocated in a virtual computer provided by Amazon's EC2 service. As for the *Cloud Server Manager* database, the H2 Database Engine [1] was used due to its small footprint, user friendly interface and JDBC API connectivity.

[1] see http://www.h2database.com/

4.2 MOFA France service description and overview

4.2.1 Workstations Description

As previously mentioned, for this case study the MOFA kit is composed by a crane robot and four workstations, namely: *Weld* workstation, *Paint* workstation, *Dry* workstation and *Drill* workstation. Each one of this workstations has a machine to operate over the product (represented by wood blocks) and positioning sensors to verify horizontal and vertical translations. The composition of each workstation is described as follows:

- ***Drill* workstation:** is composed by a machine with vertical movement and a rotary table. This table is capable of rotating 180 degrees each time to put the product under the machine head or to place it in a position to be picked or placed by the crane robot.

- ***Dry* workstation:** it uses a translation table and a set of tools (three to be exact) mounted on the head of the machine where it can chose which tool to use to perform an action on the product. The translation table is capable of moving through an axis, which allows for both the reception and the delivery of a pallet from two different sides as well as to place the pallet over the machine's head to perform an action. This machine is also capable of moving horizontally, to get near the pallet, as well as vertically, to operate on the pallet.

- ***Paint* and *Weld* workstations:** each one is composed by a machine with horizontal and vertical movements. They are supported by conveyor belts equipped with positioning sensors that are responsible for receiving the pallet from the crane robot and positioning it over the workstations head.

4.2.2 Control Organization

The new control equipment of the MOFA France kit is composed by six distributed Inico S1000 modules as represented in Figure 4.3. Each of this modules contains, in the form of services, the operations it is capable to execute regarding the machine attached to it. Therefore, as represented in Figure 4.3, each workstation is controlled by a Inico S1000 module, in the case of the *Drill* and the *Dry* workstations the module also controls their tables, the four conveyors belts are all controlled by a single module, and the same goes for the crane robot which is also controlled by a individual module. Due to the Inico S1000 limitations, the fact that the crane robot is controlled by a single Inico S1000 doesn't allow the movement of more than one axis at a time. It is important to refer that this feature is not a main subject of this dissertation, and for this reason, it was not addressed in it.

This configuration aims to represent a close approximation of the Cloud Manufacturing paradigm where each manufacturing service is separated by an individual module,

which allows a simple addition/modification/removal of a workstation without affecting the manufacturing process while allowing for the user to select only the modules he wants to use without the need to go through the others.

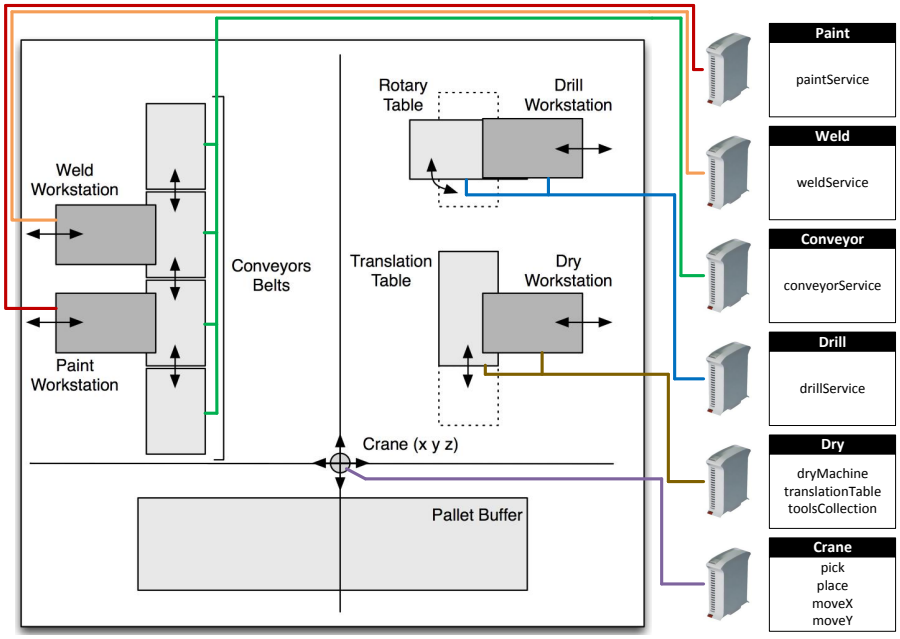


Figure 4.3: MOFA France distributed service-oriented control architecture with respective services

4.2.3 Control Configuration

The programming environment of the Inico S1000 allows for an easy definition of new Web Services, creation of Structured Text (ST) programs, link Web Services to ST programs, and definition of sending/receiving messages from the logic code. To create a new Web Service in the device it is necessary to fill up some fields as shown in Figure 4.4.

To configure a new Web Service the first step is to fill the *Service ID* and *Service types* fields. The *Service ID* field is the unique identifier of the Web Service and therefore is mandatory. The *Service types* is a list of types that describes this type of service. Service types allows to categorize Web Services which is useful to organize and work with all the Web Services in a particular network. With this two fields filled, it is required to add messages to the Web Service.

There are three types of messages that can be added to a Web Service:

- **Input messages:** messages that are sent from a PC application to the Inico. These messages convey commands to execute a particular action, or can also be used to transfer configuration data. They can also be used to request data, such as the state of a process or the value of a measured parameter. A response message from the Inico to the PC application can be configured, or left blank if not used.

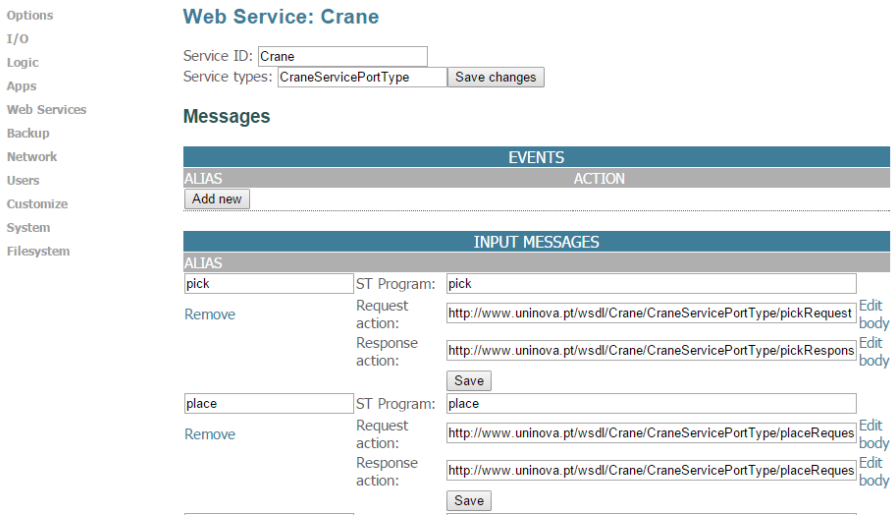


Figure 4.4: Inico S1000 programming environment - example of the configuration of a service

- **Output messages:** messages sent from the Inico to a PC application. As with the Input messages, these messages may or may not use a response message.
- **Event messages:** these are messages that are sent from the Inico to one or more PC applications. Typically, these messages report events such as the completion of a process, a significant change in a process parameter, or a fault/error/warning condition. In order to receive a copy of these messages, the PC applications must *subscribe* to the Inico, so that an internal list of interested applications can be maintained. The number of subscribed applications may vary from 0 to many.

These types of messages can be defined with four fields, some of them are not used by all types, namely:

- **Alias:** used by all three types of message, is used as a reference to this message from ST logic.
- **Program name:** used only by Input messages, defines the name of the ST program to be executed when this message is received.
- **Request Action:** used by all three types of message, is used to identify the type of this message.
- **Response Action:** used by Input and Output messages but with different meanings. In the case of the Input messages, the response action is used to identify the type of the response message to be generated. In the case of the Output message, it is used to identify the type of the response message to be received.

The specification of the message XML structure of each message is a crucial step because this is what defines whether the incoming message is valid or not. This structure is

defined by the user through the programming environment of the Inico S1000 as shown in the example of Listing 4.1.

Listing 4.1: Inico S1000 programming environment - example of Input message in XML-based format

```
<moveYRequest xmlns="http://www.uninova.pt/wsdl/Crane">
  <moveYInput xmlns="http://www.uninova.pt/wsdl/Crane">$YcoordReq</moveYInput>
</moveYRequest>
```

When an Input message is received the specified ST program will be executed. If any variable data was sent with the message, this data will be available from the string variables configured in the XML message structure. In addition to the logic executed, the ST program must send back a response message using the *WS_RESPOND* command, even if no SOAP response is configured. When a ST program needs to publish a message to every PC applications subscribed to a specific service, the *WS_PUBLISH* command is used to generate and send a message to the subscribed PC applications. In Listing 4.2 is possible to see an example of a ST program that can be invoked from a Web Service with the use of an Input message (in this case, the *pick* Input message as shown in Figure 4.4).

Listing 4.2: Inico S1000 programming environment - example of ST program

```
PROGRAM pick (** Edit new program name **)

    Z_Down := true;
    wait_until(Sensor_Z_Down = false);
    Z_Down := false;
    Magnet := true;
    Z_Up := true;
    wait_until(Sensor_Z_Up = false);
    Z_Up := false;

    pick_response := 'Position␣Reached';
    ws_respond(pick);
END_PROGRAM
```

For a service to be discovered by a PC application it is necessary to describe the service with the use of WSDL. For each service presented in an Inico module, a WSDL must be created so that DPWS implementations can understand how to communicate with the services allocated in the module. The WSDL file must have the same names declared in the configuration of each service as described above. In Appendix 1 is presented the WSDL file capable of describing the service presented in Figure 4.4.

4.2.4 Description of the Operations

In Table 4.1 a list of operations implemented to control the MOFA kit is presented. Each of this operations interact with the sensors and actuators of the MOFA kit to execute an action. The operations interact with the user by inputs and outputs where the user can

specify the operation details by giving the desired inputs. In response the operation will be executed and the user notified when it is complete.

Table 4.1: Description of the service operations

Device Name	Service Name	Operation Name	Inputs	Description
Paint	paintService	paint	paintTime [0..10000]	enables paint operation over a pallet
Weld	weldService	weld	weldTime [0..10000]	enables weld operation over a pallet
Conveyor	conveyorService	movePiece	id [1..n] action [paint,weld]	transports a pallet from one belt to another
		isPositionFree	type [entry,exit]	verifies if if the type belt is free
Drill	drillService	drill	drilTime [0..10000]	enables drill operation over a pallet
Dry	dryMachine	dry	dryTime [0..10000]	enables dry operation over a pallet
	translationTable	switchTable	drySide [left, right]	translate the table to the left or right side
	toolsCollection	changeTool	toolNumber [1,2,3]	select one of the three tools to dry a pallet
Crane	Crane	pick	no input	crane robot picks a pallet in the current position
		place	no input	crane robot places a pallet in the current position
		moveX	pos [0..72]	move the crane robot to the indicated position in the X axis
		moveY	pos [0..67]	move the crane robot to the indicated position in the Y axis

4.3 Deployer

The *Deployer* is the component/resource that sets on the manufacturing company side. It is responsible for the detection, operation and deploying of the devices and services existent in the manufacturing company. The *Deployer* is composed by a perceptive GUI

(see Figure 4.5) that allows the operator to search for devices and services in the network as well as visualize their metadata, order them to execute operations, subscribe to events or deploy them in the cloud. The GUI is also composed by a log where the operator can verify what the *Deployer* is doing and errors that occurred.

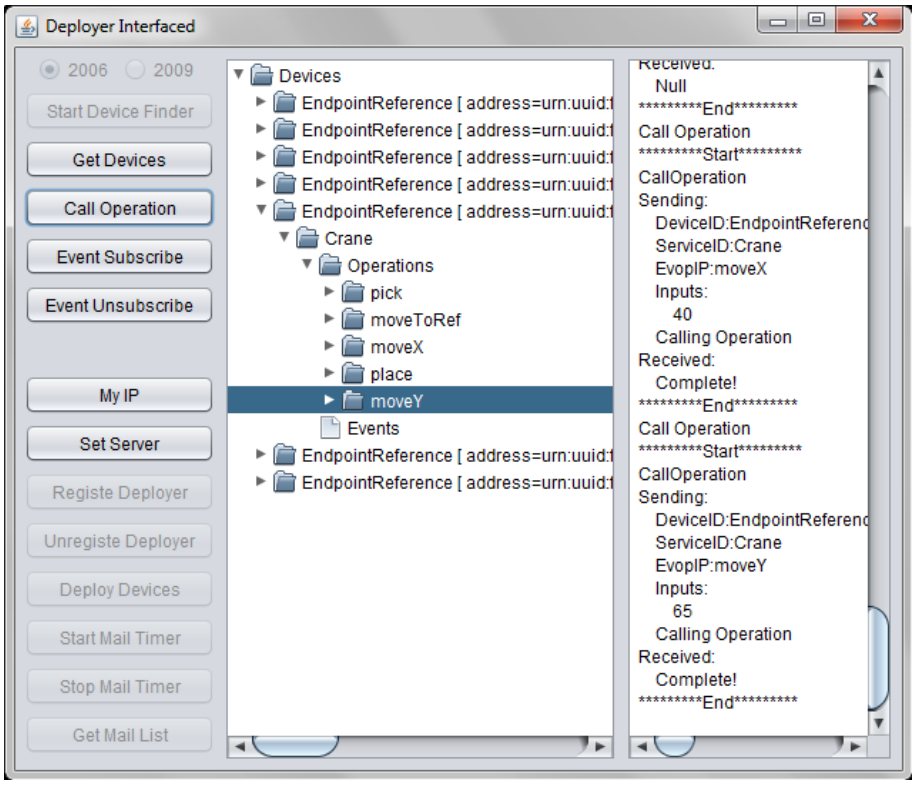


Figure 4.5: *Deployer* GUI

To do so, the *Deployer* was developed in Java with the use of the JMEDS framework and the Apache CXF services framework. The use of the JMEDS framework supports a network broadcast discovery which allowed to search and interact with devices supporting DPWS by exploiting the WS-Discovery protocol embedded in the DPWS protocol stack. As for the Apache CXF services framework, its capability of dynamically creating and calling Web Services enabled the communications with the *Cloud Server Manager*.

The *Deployer* was developed in a way that the operator can chose to deploy the manufacturing devices and still operate them if he needs to, therefore the manufacturing company continues its normal operations while at the same time executes the clients requests. In Figure 4.5 it is possible to see a set of buttons that are used by the operator not only to operate the devices connected to the network, but also to deploy them in the cloud. Among this buttons it is possible to see a *Set Server* button as well as a *Registe Deployer* button, which allows the operator to register the *Deployer* in a chosen server or a list of servers where clients can find the deployed manufacturing devices and operate them as they please.

As discussed in Section 3.1 the *Deployer* is composed by four main modules (*Device*

Explorer, *Device Virtualization*, *Device Repository* and *Device Handler*), a Interface (see Figure 4.5), and a Listener.

4.3.1 Device Explorer

As the name indicates, the *Device Explorer* module is responsible to search the network for available *DPWS*-enabled devices. This module uses the JMEDS framework to retrieve not only the available devices but also their metadata. JMEDS offers a *SearchManager* class to get hold of devices and services on the network. This class provides tools to search local and remote devices and services with or without a set of search criteria, and obtaining references to devices/services with known endpoint addresses. With the implementation of the *SearchCallback* interface it is possible to receive the discovered device or service during the search by the means of asynchronous notifications and therefore assemble the *Device Explorer*.

One constrain found in the JMEDS framework *Search Manager* was the fact that when a device is discovered, the *SearchCallback* doesn't return a complete description of the device. This means that if someone wants to call an operation of the device, he first needs to search for the device and specifically request for the operation before he can execute something on it. This represented a problem due to the connection timeout that sometimes occurred and represented a complete lost of the device making it temporarily unavailable. To solve this problem, when a device is discovered, the *Device Explorer* immediately requests for all the data contained in the device and stores it, so that when someone needs to execute an operation it can directly request the device to execute the requested operation instead of first asking the device for the operation and then ask to execute it, thus avoiding the risk of connection timeout.

4.3.2 Device Virtualization

This module transforms each device returned by the *Device Explorer* into two types of representations, one, as described before, to avoid the connection timeout problem and to directly execute an operation instead of first request the operation and then request its execution, and another so that it can be stored and passed to other applications without the need of having the JMEDS framework. This way, the device virtual representation becomes easier to understand and requires less complexity to interact with. This allows users to build their own *Client* applications without the need of adding the JMEDS framework since they wont be using their full functionalities.

To achieve this virtualization a Java class was developed containing the relevant device metadata and keeping its topology intact. This allows for an easy visualization of the device among with its own metadata, hosted services, operations and event sources.

4.3.3 Device Repository

The *Device Repository* stores the virtualizaton of the founded devices so that they can be both displayed to the operator and deployed in the cloud without the need of being constantly searching for devices and also avoid the sporadic connection failures. The devices are stored in the run-time application with the use of a *Hashmap* containing the virtualization of each device.

4.3.4 Device Handler

The *Device Handler* is the module used to interact with the devices. With the use of the JMEDS framework, this module is able to order the devices to execute operations, register to events, and handle the responses from the operations and events as well as to manage the requests. To do so, when the *Device Handler* receives a request it searches the *Device Repository* for the corresponding action, the *Device Repository* returns the details on how to call the requested action, through the JMEDS framework the *Device Handler* uses this information to contact with the responsible device to execute the requested operation. When the requested operation is completed, the *Device Handler* receives a message from the device, which is then passed, by the *Device Handler*, to who made the original request (the *Deployer* itself or the *Cloud Server Manager*).

4.4 Cloud Server Manager

As previously mentioned, the *Cloud Server Manager* is allocated in a cloud computing platform, more precisely the Amazon Elastic Compute Cloud. In this cloud was installed an Ubuntu platform offered by the AWS, where the *Cloud Server Manager* sits on. This platform is composed by a minimum Ubuntu image only containing the essential for the operating system to work.

For the *Cloud Server Manager* an interface (see Figure 4.6) was developed to offer basic control over the *Cloud Server Manager* since this component/resource is located in the cloud with minimum access to it, therefore this interface only allows for the creation/-clean of the database where the device, deployer, client and requested information is stored, definition of the server properties, initialization of the server so that others (*Deployer*, *Client UI*) can connect to him and also a shut down method to disable connections to the *Cloud Manager Server*.

The *Cloud Server Manager* is composed by two web services implemented with the Apache CXF and a local database made with the use of H2 Database Engine. Each web service was designed with service specific functions according to the component/resource destined to (*Deployer* or *Client UI*).

The *Deployer Services* Web Service allows the *Deployer* component/resource to register himself in the *Cloud Server Manager*, which allows him to deploy its devices and services so that others can interact with him. The *Deployer Services* also allows for the *Deployer*

Figure 4.6: *Cloud Server Manager* GUI

to consult its mail list where all the requests are stored. By consulting this mail list the *Deployer* can collect the operations requested by the *Client UI* as well as its inputs and execute them, and when the execution is done, the *Deployer* uses the *post_mail* service from the *Deployer Services* to post the outputs of the executed operations in the *Cloud Server Manager* which will transmit them to the destined *Client UI*.

The *Client Services* Web Service is composed by six services, that allow the *Client UI* to get the devices deployed in the *Cloud Server Manager* and request this devices to execute operations or subscribe to their events. The Web Service also contains the register and unregister functionalities, as well as, the get mail list functionality where the *Client UI* retrieves from the *Cloud Server Manager* the outputs returned by the operations or events subscribed.

The local database was developed to store the devices deployed by the *Deployer* in an understandable way, focusing on the essentials and removing the need to use the JMEDS framework. In Figure 4.7 it is possible to see the Entity Diagram of the *Cloud Server Manager*. During its development, was found the need to request asynchronous operations, meaning that neither the *Cloud Server Manager* or the *Client UI* should be blocked waiting for a response. Another need, was to have a history record for both debug and security reasons. Therefore in this database, a mailing list table was introduced where the requests are stored with a triple meaning, firstly, so that the *Deployer* can interact with the *Client UI* by consulting this mail list table to get the requested executions as well as their inputs, secondly, so that the *Client UI* can consult the mail list table to see if the requested executions are already completed and if they returned any output, and thirdly, to keep a record of the executed operations for debug and security reasons.

4.5 Client UI

As previously mentioned, the *Client UI* was developed to mimic the *Deployer* in the user point of view. To achieve this, an interface similar to the *Deployer* was developed as shown in Figure 4.8, with the objective to give to the users the look and feel of being next to the device itself and operating it just like an operator would do with the use of the

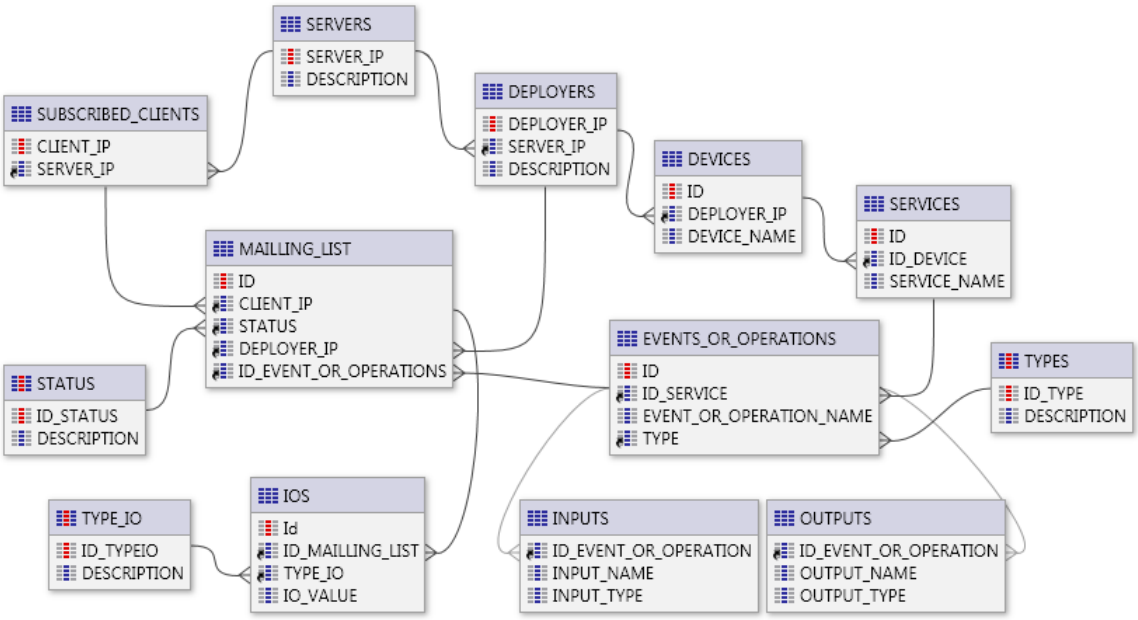


Figure 4.7: *Cloud Server Manager* Entity Diagram

Deployer component/resource.

Like the *Deployer*, the *Client UI* can visualize the Devices as well as their metadata, request the execution of operations and both subscribe and unsubscribe to events. The only difference between the two is that while the *Deployer* directly calls the device in question, the *Client UI* needs to pass through the *Cloud Server Manager* and the *Deployer*, only then will the request reach the device. For this to happen the *Client UI* first needs to register himself in the *Cloud Server Manager* so that it can have access to the mailing list where the outputs of the requested actions are published.

4.6 Tests and Results

To test the correct functioning of the architecture as well as the implemented components, a four stage process was implemented:

1. Use only the *Deployer* to execute a task locally.
2. Connect both the *Deployer* and the *Client UI* to the *Cloud Server Manager* and verify if they were properly registered in the *Cloud Server Manager*.
3. Deploy the devices collected by the *Deployer* into the *Cloud Server Manager* and use the *Client UI* to retrieve them.
4. Use the *Client UI* to execute a service from one of the retrieved devices.

Since the *Deployer* uses the JMEDS framework, when launching the *Deployer* the first thing to do is to start the framework by selecting the DPWS protocol used by the devices

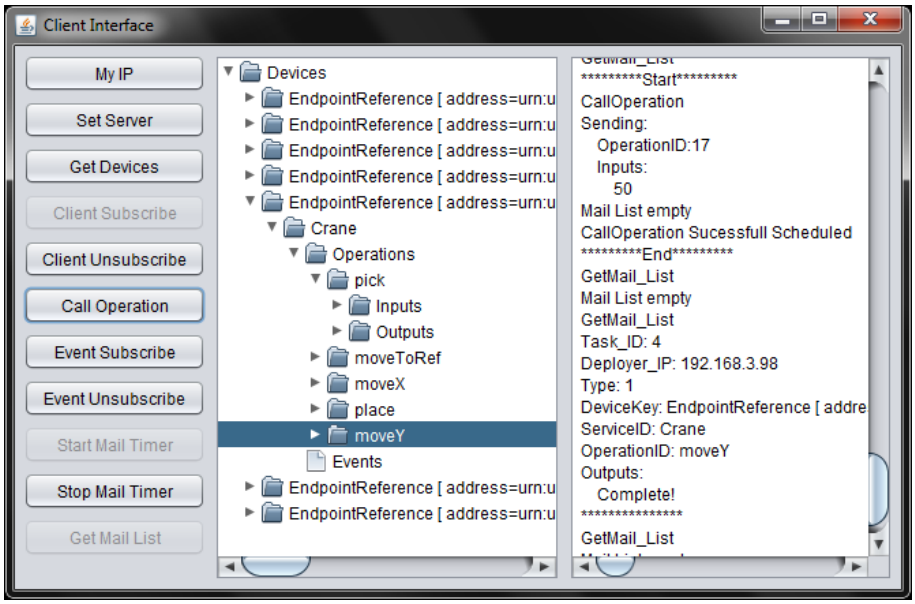


Figure 4.8: *Client UI* GUI

in question (in this case the INICO devices use 2006 DPWS protocol), and by clicking the *Start Device Finder* button. This will start the framework which will allow the search and interaction with the devices, and will trigger the timer that constantly searches the network for devices and adds them to the *Device Repository* when a new device is found. With this ready the first step to execute a task is to select a device containing the desired task, therefore by clicking the *Get Devices* button, the devices stored in the *Device Repository* are displayed to the operator so that he can select the desired operation. Using the devices tree, the operator can select the desired operation and by clicking the *Call Operation Button* a new window is shown so that the operator can provide the inputs for the operation (in case the operation doesn't require inputs this window will not be shown and the operation will start immediately). Once the inputs are provided the *Deployer* sends the request to the chosen device, which starts the execution of the operation and returns a response when it's finished. For this test the *Crane* device was selected to execute the *moveX* operation with the input "0". This made the crane move to the zero position of its X axis and then retrieve the following message "Received: Complete!". With this outcome the first stage of the process was considered a success.

For the second stage, first the *Cloud Server Manager* needs to be configured and initiated. For that it first needs to know its public and local IP addresses, which can be configured by clicking the *Set Server IP* button where a window will pop up and the user can input this addresses. When this configuration is done the *Cloud Server Manager* can be initiated by clicking the *Start Server* button that will allow others to connect to him through its web services (*Deployer Services* and *Client Services*). By receiving the messages: "Deployer Server Started at: http://54.77.151.44:8081/DeployerServices?wsdl" and "Client Server Started at: http://54.77.151.44:8082/ClientServices?wsdl"; the confirmation is made that

both web services where properly initiated and are up and running, ready to receive requests.

With the *Cloud Server Manager* up and running, the *Deployer* and *Client UI* can now connect to him. Just like the *Cloud Server Manager* they first need to configure their IP addresses so that the *Cloud Server Manager* can distinguish them from other *Deployer*s and *Client UI*s. This is made by clicking the *My IP* button where, like in the *Cloud Server Manager*, a new window will appear for the user to input its public IP address (this is valid for both *Deployer* and *Client UI*). Once this step is made, the web service url needs to be set so that each component connects to the right web service, i.e. the *Deployer* connects to the *Deployer Services* and the *Client UI* connects to the *Client Services*. By clicking the *Set Server* button, the user can input the url to the desired service. This url is composed by the *Cloud Server Manager* public IP address, its port and the web service name, for example "*http://54.77.151.44:8081/DeployerServices?wsdl*". With the configuration done the only step missing is the registration in the *Cloud Server Manager*. When clicking the *Registe Deployer* button (*Client Subscribe* in the case of the *Client UI*) the component sends a request to the *Cloud Manager Server*, it was possible to see that the request reached the *Cloud Server Manager* due to the message "registerDeployer - Received request" ("ClientSubscribe - Received request" in the case of the *Client UI*) shown in the *Cloud Server Manager* interface. The *Cloud Server Manager* then retrieved a confirmation message saying: "Sucessfull Registration" which confirmed that the communications are up and running classifying the second stage of the tests as successful.

To deploy the devices, the user, on the *Deployer* side, only needs to click the *Deploy Devices* button, which send to the *Cloud Server Manager* the devices contained in the Device Repository. As confirmation of its proper execution the message "Deployement Sucessfull" was obtained.

On the *Client UI* side, by clicking the *Get Devices* button, the *Cloud Server Manager* sends a list of all the devices stored in its database which the *Client UI* displays in its interface in the form of a tree, so that the user can see all the devices retrieved as well as the list of services and operations contained in each device, thus settling also stage three as successful.

To finalize the test, the *Client UI* will be used to request the execution of an operation of one of the retrieved devices. For this step, two options are provided: use the timer to periodically retrieve the mail list that contains the requests and responses of operations from the *Cloud Server Manager* or retrieve the mail list by hand when the user wants. For this test the timer was used on both *Deployer* and *Client UI*, and for that the button *Start Mail Timer* first needs to be clicked on both *Deployer* and *Client UI* (note that it is not mandatory that both components use the timer, one can be using the timer while the other is not).

Since the *Client UI* was developed to mimic the *Deployer*, the process of calling an operation is almost the same. Therefore the user starts by selecting the desired operation from the previously retrieved device list. With the operation selected, by clicking the

Call Operation button the user is allowed to provide the inputs for the operation (if the operation has inputs). For this test the *Crane* device was selected to execute the *moveY* operation whit the input "0", which will make the crane move to the zero position of its Y axis. With this information filled the *Client UI* then sends a request containing this information to the *Cloud Server Manager* mail list, so that the operation corresponding *Deployer* can go fetch the request and execute the requested operation. This step could be confirmed with the message "CallOperation Sucessfull Scheduled". When the *Deployer* timer is triggered, it contacts the *Cloud Server Manager* to get the requested operations. With this the *Deployer* starts to perform the requested operations and whenever one operation is completed it publishes the result in the *Cloud Server Manager* so that the *Client UI* can consult its output.

The confirmation for this final stage was obtained when the *Crane* moved to the requested position, and when the message coming from the *Cloud Server Manager* switched from "Mail List empty" to the previously sent message request, with the addition of the outputs returned by the executed operation so that the user knows to which operation request does the outputs correspond to.

For the execution of this test the MOFA kit was used as a physical simulator of a Cloud Manufacturing environment where manufacturing tasks are deployed as services that can be collected by the *Deployer* which was installed in a computer set next to the MOFA kit in a room located in Almada, Portugal. The *Cloud Server Manager* was installed in the Amazons servers located in Ireland and the *Client UI* in a home personal computer located in Lisbon, Portugal. Therefore, all three components/resources where placed in different geographical areas thus proving the well functioning of the deployment platform as well as for what it was planned.

5

Conclusions and Future Work

5.1 Conclusions

With the use of the deployment platform, manufacturing services can be spread worldwide, allowing the consumers to control or even create their own products. By combining the services of several manufacturers in the same platform, the user frees himself from the constrains of selecting services from just one manufacturer, thus new innovative services can be achieved with the use of several manufacturers services, giving total freedom for the user to select what he really wants.

The presented infrastructure provides a cloud-based manufacturing system where available manufacturing resources are virtualized as cloud entities and which capabilities/functionalities are exposed as services for enabling the creation of processes by the composition of this capabilities/functionalities. During this work some fundamental pillars of the Cloud Manufacturing paradigm have been approached, namely: virtualization access, deployment into cloud and service encapsulation of physical manufacturing resources.

With the presented work, the feasibility of the Cloud Manufacturing paradigm has been proved when supported by intelligent/smart devices. Moreover, the implemented infrastructure provides a normalized layer where the typical heterogeneity of the automation domain is hidden into a generic semantic representation.

Furthermore, since the solution is supported by open web standards it leaves the door open for the integration of more internet-based solutions. However, more extensive validations, as well as improvements are required to turn this solution into a proper, reliable and secure solution to address the Cloud Manufacturing paradigm.

5.2 Future Work

As it was referred above, several improvements should be done before this could be considered a valid solution to address the Cloud Manufacturing paradigm, namely:

- Implementation of a service composition solution so that a user can create a composition of services from different manufacturers in both asynchronous and synchronous ways. This should be done by creating a service broker where the call of operations is delegated by the *Cloud Server Manager*.
- Improve the *Cloud Service Manager* web services to allow the reception of a service composition from a user instead of receiving one by one.
- Integration of graphical tools for service composition (BPMN, BPEL, etc.).
- The semantic representation can be significantly improved by introducing ontologies (OWL, OWL-S, etc.).
- Improve the communication between device and *Deployer* so that the *Deployer* has the capability to call several devices at the same time, but also be aware of service compositions overlaps.
- Make the communications both secure and reliable, maybe with the use of encryption, since the communications are made through the internet.
- Implementation of a login access to control the requests.

5.3 Scientific Contributions

This work resulted in three scientific contributions published in:

1. Jassbi, J., di Orio, G., Barata, D., and Barata, J. (2014). The impact of cloud manufacturing on supply chain agility. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*
2. Di Orio, G., Barata, D., Rocha, A., and Barata, J. (2015). A cloud-based infrastrucutre to support manufacturing resources composition. In *6th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS'15)*
3. Rocha, A., Barata, D., Di Orio, G., Santos, T., and Barata, J. (2015). PRIME as a generic agent based framework to support pluggability and reconfigurability using different technologies. In *6th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS'15)*

Bibliography

[Apache, 2009] Apache (2009). Apache CXF: An open-source services framework.

[Armbrust et al., 2010] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4):50–58.

[Babiceanu and Chen, 2006] Babiceanu, R. F. and Chen, F. F. (2006). Development and applications of holonic manufacturing systems: A survey. *Journal of Intelligent Manufacturing*, 17(1):111–131.

[Barata et al., 2008] Barata, J., Camarinha-Matos, L., and Cândido, G. (2008). A multiagent-based control system applied to an educational shop floor. *Robotics and Computer-Integrated Manufacturing*, 24(5):597–605.

[Barata et al., 2006] Barata, J., Santana, P., and Onori, M. (2006). Evolvable assembly systems: a development roadmap.

[Barry, 2003] Barry, D. K. (2003). *Web Services, Service-Oriented Architectures: The Savvy Manager's Guide*. Morgan Kaufmann.

[Bell, 1962] Bell, D. (1962). *The End of Ideology: On the Exhaustion of Political Ideas in the Fifties : with "The Resumption of History in the New Century"*. Harvard University Press.

[Bellwood et al., 2002] Bellwood, T., Capell, S., Clement, L., Colgrave, J., Dovey, M., Feygin, D., Kochman, A., Macias, P., Novotny, M., Paolucci, M., and others (2002). Universal description, discovery and integration specification (UDDI) 3.0. *Online: http://uddi. org/pubs/uddi-v3. 00-published-20020719. htm*.

[Box et al., 2000] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). *Simple object access protocol (SOAP) 1.1*.

[Bray et al., 1998] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (1998). Extensible markup language (XML). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*.

[Bronsted et al., 2010] Bronsted, J., Hansen, K., and Ingstrup, M. (2010). Service composition issues in pervasive computing. *IEEE Pervasive Computing*, 9(1):62–70.

[Browne et al., 1988] Browne, J., Harhen, J., and Shivnan, J. (1988). *Production management systems: a CIM perspective*. Addison-Wesley Wokingham.

[Bucchiarone et al., 2007] Bucchiarone, A., Melgratti, H., and Severoni, F. (2007). Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE'07)*.

[Camarinha Matos et al., 1995] Camarinha Matos, L. M., Pinheiro Pita, H., Rabelo, R., and Barata, J. (1995). Towards a taxonomy of CIM activities. *International Journal of Computer Integrated Manufacturing*, 8(3):160–176.

[Candido et al., 2011] Candido, G., Colombo, A., Barata, J., and Jammes, F. (2011). Service-oriented infrastructure to support the deployment of evolvable production systems. *IEEE Transactions on Industrial Informatics*, 7(4):759–767.

[Candido et al., 2010] Candido, G., Jammes, F., de Oliveira, J., and Colombo, A. (2010). SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications. In *2010 8th IEEE International Conference on Industrial Informatics (INDIN)*, pages 598–603.

[Chou, 2010] Chou, T. (2010). *Introduction to Cloud Computing*. Cloudbook.

[Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., and others (2001). *Web services description language (WSDL) 1.1*.

[Christensen, 1994] Christensen, J. H. (1994). Holonic manufacturing systems: initial architecture and standards directions. *Proc 1st Euro Wkshp on Holonic Manufacturing Systems*.

[Cândido et al., 2009] Cândido, G., Barata, J., Colombo, A. W., and Jammes, F. (2009). SOA in reconfigurable supply chains: A research roadmap. *Engineering Applications of Artificial Intelligence*, 22(6):939–949.

[Cândido, 2013] Cândido, G. M. (2013). *Service-oriented architecture for device lifecycle support in industrial automation*. PhD thesis. Dissertação para obtenção do Grau de Doutor em Engenharia Electrotécnica e de Computadores Especialidade: Robótica e Manufactura Integrada.

[Colombo and Karnouskos, 2009] Colombo, A. W. and Karnouskos, S. (2009). Towards the factory of the future: A service-oriented cross-layer infrastructure. *ICT Shaping the World: A Scientific View. European Telecommunications Standards Institute (ETSI), John Wiley and Sons*, 65:81.

[Consortium and others, 2004] Consortium, W. W. W. and others (2004). Web services glossary. *Retrieved May*, 31:2009.

[De Masi, 2000] De Masi, D. (2000). *A sociedade Pós Industrial*. SENAC, São Paulo.

[Di Orio, 2013] Di Orio, G. (2013). *Adapter module for self-learning production systems*. PhD thesis. Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica, Sistemas e Computadores.

[Di Orio et al., 2015] Di Orio, G., Barata, D., Rocha, A., and Barata, J. (2015). A cloud-based infrastrucutre to support manufacturing resources composition. In *6th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS'15)*.

[Dustdar and Schreiner, 2005] Dustdar, S. and Schreiner, W. (2005). A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30.

[Erl, 2005] Erl, T. (2005). *Service-oriented architecture: concepts, technology, and design*. Pearson Education India.

[Ford and Crowther, 1988] Ford, H. and Crowther, S. (1988). TODAY AND TOMORROW.

[Frei et al., 2007] Frei, R., Barata, J., and Onori, M. (2007). Evolvable production systems context and implications. In *IEEE International Symposium on Industrial Electronics, 2007. ISIE 2007*, pages 3233–3238.

[Goldman et al., 1995] Goldman, S. L., Nagel, R. N., and Preiss, K. (1995). *Agile competitors and virtual organizations: strategies for enriching the customer*, volume 8. Van Nostrand Reinhold New York.

[Gross, 1996] Gross, D. (1996). *Forbes greatest business stories of all time*. J. Wiley & Sons.

[Hajarnavis and Young, 2008] Hajarnavis, V. and Young, K. (2008). An assessment of PLC software structure suitability for the support of flexible manufacturing processes. *IEEE Transactions on Automation Science and Engineering*, 5(4):641–650.

[Hames, 1994] Hames, R. D. (1994). *The management myth: Exploring the essence of future organisations*. Business & Professional Publishing.

[Jammes et al., 2005] Jammes, F., Mensch, A., and Smit, H. (2005). Service-oriented device communications using the devices profile for web services. In *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing*, MPAC '05, pages 1–8, New York, NY, USA. ACM.

[Jammes and Smit, 2005] Jammes, F. and Smit, H. (2005). Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1(1):62–70.

[Jassbi et al., 2014] Jassbi, J., di Orio, G., Barata, D., and Barata, J. (2014). The impact of cloud manufacturing on supply chain agility. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*.

[Josuttis, 2007] Josuttis, N. M. (2007). *SOA in Practice: The Art of Distributed System Design*. "O'Reilly Media, Inc.".

[Karnouskos et al., 2012] Karnouskos, S., Colombo, A., Bangemann, T., Manninen, K., Camp, R., Tilly, M., Stluka, P., Jammes, F., Delsing, J., and Eliasson, J. (2012). A SOA-based architecture for empowering future collaborative cloud-based industrial automation. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 5766–5772.

[Karnouskos et al., 2010] Karnouskos, S., Savio, D., Spiess, P., Guinard, D., Trifa, V., and Baecker, O. (2010). Real-world service interaction with enterprise systems in dynamic manufacturing environments. In Benyoucef, L. and Grabot, B., editors, *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, Springer Series in Advanced Manufacturing, pages 423–457. Springer London.

[Kavantzas et al., 2005] Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., and Barreto, C. (2005). Web services choreography description language version 1.0. *W3C candidate recommendation*, 9.

[Koestler, 1968] Koestler, A. (1968). *The Ghost in the Machine*, volume xvi. Macmillan, Oxford, England.

[Komoda, 2006] Komoda, N. (2006). Service oriented architecture (SOA) in industrial systems. In *2006 IEEE International Conference on Industrial Informatics*, pages 1–5.

[Koren, 2010] Koren, Y. (2010). *The Global Manufacturing Revolution: Product-Process-Business Integration and Reconfigurable Systems*. John Wiley & Sons.

[Koren et al., 1999] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., and Van Brussel, H. (1999). Reconfigurable manufacturing systems. *CIRP Annals - Manufacturing Technology*, 48(2):527–540.

[Kornhauser, 1959] Kornhauser, W. (1959). The politics of mass society. *Free Press New York*.

[Krafzig et al., 2005] Krafzig, D., Banke, K., and Slama, D. (2005). *Enterprise SOA: Service-oriented Architecture Best Practices*. Prentice Hall Professional.

[Leitao, 2004] Leitao, P. J. P. (2004). *An agile and adaptive holonic architecture for manufacturing control*. PhD thesis, University of Porto.

[Licklider, 1963] Licklider, J. C. R. (1963). Topics for discussion at the forthcoming meeting, memorandum for: Members and affiliates of the intergalactic computer network.

[Liu et al., 2011] Liu, Q., Gao, L., and Lou, P. (2011). Resource management based on multi-agent technology for cloud manufacturing. In *2011 International Conference on Electronics, Communications and Control (ICECC)*, pages 2821–2824.

[Makino and Tominaga, 1995] Makino, H. and Tominaga, M. (1995). Estimation of production rate in flexible assembly systems. *CIRP Annals - Manufacturing Technology*, 44(1):7–10.

[Marinos and Briscoe, 2009] Marinos, A. and Briscoe, G. (2009). Community cloud computing. In Jaatun, M. G., Zhao, G., and Rong, C., editors, *Cloud Computing*, number 5931 in Lecture Notes in Computer Science, pages 472–484. Springer Berlin Heidelberg.

[Mehrabi et al., 2000] Mehrabi, M. G., Ulsoy, A. G., and Koren, Y. (2000). Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing*, 11(4):403–419.

[Mehrabi et al., 2002] Mehrabi, M. G., Ulsoy, A. G., Koren, Y., and Heytler, P. (2002). Trends and perspectives in flexible and reconfigurable manufacturing systems. *Journal of Intelligent Manufacturing*, 13(2):135–146.

[Mell and Grance, 2009] Mell, P. and Grance, T. (2009). Perspectives on cloud computing and standards. *National Institute of Standards and Technology (NIST), Information Technology Laboratory*.

[Mitchell, 1991] Mitchell, Jr., F. H. (1991). *CIM Systems: An Introduction to Computer-integrated Manufacturing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Natis, 2003] Natis, Y. V. (2003). *Service-oriented architecture scenario*.

[Neves and Barata, 2009] Neves, P. and Barata, J. (2009). Evolvable production systems. In *IEEE International Symposium on Assembly and Manufacturing, 2009. ISAM 2009*, pages 189–195.

[Okino, 1993] Okino, N. (1993). Bionic manufacturing systems. In *Conference on Flexible Manufacturing Systems, Past, Present-Future (Ed: J. Peklenik), Ljubljana: Faculty of Mechanical Engineering*, pages 73–95.

[Onori, 2002] Onori, M. (2002). Evolvable assembly systems : A new paradigm? Evolvable Assembly Systems : A New Paradigm?

[Onori et al., 2006] Onori, M., Barata, J., and Frei, R. (2006). Evolvable assembly systems basic principles. In *Information Technology For Balanced Manufacturing Systems*, number 220 in IFIP International Federation for Information Processing, pages 317–328. Springer US.

[Papazoglou and Heuvel, 2007] Papazoglou, M. P. and Heuvel, W.-J. v. d. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415.

[Peltz, 2003] Peltz, C. (2003). Web services orchestration and choreography. *Computer*, 36(10):46–52.

[Pine, 1999] Pine, B. J. (1999). *Mass Customization: The New Frontier in Business Competition*. Harvard Business Press.

[Ranky, 1990] Ranky, P. G. (1990). *Flexible manufacturing cells and systems in CIM*. CIMware.

[Ribeiro and Barata, 2011] Ribeiro, L. and Barata, J. (2011). Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging IT based production paradigms. *Computers in Industry*, 62(7):639–659.

[Ribeiro et al., 2008] Ribeiro, L., Barata, J., and Mendes, P. (2008). MAS and SOA: Complementary automation paradigms. In Azevedo, A., editor, *Innovation in Manufacturing Networks*, number 266 in IFIP – The International Federation for Information Processing, pages 259–268. Springer US.

[Ribeiro et al., 2011] Ribeiro, L., Candido, G., Barata, J., Schuetzy, S., and Hofmann, A. (2011). IT support of mechatronic networks: A brief survey. In *2011 IEEE International Symposium on Industrial Electronics (ISIE)*, pages 1791–1796.

[Rocha et al., 2015] Rocha, A., Barata, D., Di Orio, G., Santos, T., and Barata, J. (2015). PRIME as a generic agent based framework to support pluggability and reconfigurability using different technologies. In *6th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS'15)*.

[Scheer, 1991] Scheer, A.-W. (1991). *CIM Computer Integrated Manufacturing - Towards the Factory of the Future*. Springer, Berlin.

[Sousa, 2014] Sousa, C. F. G. d. (2014). *Service-oriented infrastructure to support the control, monitoring and management of a shop floor system*. PhD thesis. Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores.

[Tao et al., 2013] Tao, F., LaiLi, Y., Xu, L., and Zhang, L. (2013). FC-PACO-RM: A parallel method for service composition optimal-selection in cloud manufacturing system. *IEEE Transactions on Industrial Informatics*, 9(4):2023–2033.

[Tao et al., 2011] Tao, F., Zhang, L., Venkatesh, V. C., Luo, Y., and Cheng, Y. (2011). Cloud manufacturing: a computing and service-oriented manufacturing model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 225(10):1969–1976.

[Tharumarajah, 1996] Tharumarajah, A. (1996). Comparison of the bionic, fractal and holonic manufacturing system concepts. *International Journal of Computer Integrated Manufacturing*, 9(3):217–226.

[Ueda, 1992] Ueda, K. (1992). A concept for bionic manufacturing systems based on DNA-type information. In *Proceedings of the IFIP TC5 / WG5.3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing*, pages 853–863, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

[Upton, 1992] Upton, D. M. (1992). A flexible structure for computer-controlled manufacturing systems. *Manufacturing Review*, 5(1):58–74.

[Van Brussel et al., 1998] Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., and Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37(3):255–274.

[Vincent Wang and Xu, 2013] Vincent Wang, X. and Xu, X. W. (2013). An interoperable solution for cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 29(4):232–247.

[Waldner and Duffin, 1992] Waldner, J.-B. and Duffin, W. (1992). *CIM, principles of computer-integrated manufacturing*. Wiley Chichester.

[Wang, 2013] Wang, L. (2013). Machine availability monitoring and machining process planning towards cloud manufacturing. *CIRP Journal of Manufacturing Science and Technology*, 6(4):263–273.

[Wang and Xu, 2013] Wang, X. V. and Xu, X. W. (2013). ICMS: A cloud-based manufacturing system. In Li, W. and Mehnen, J., editors, *Cloud Manufacturing*, Springer Series in Advanced Manufacturing, pages 1–22. Springer London.

[Womack et al., 1990] Womack, J. P., Jones, D. T., and Roos, D. (1990). *Machine that Changed the World*. Scribner.

[WS4D, 2012] WS4D (2012). Stack: WS4d-JMEDS (java) | web services for devices.

[Wu et al., 2013] Wu, D., Greer, M. J., Rosen, D. W., and Schaefer, D. (2013). Cloud manufacturing: Strategic vision and state-of-the-art. *J. Manuf. Syst. Available at: http://www. sciencedirect. com/science/article/pii S*.

[Xu, 2012] Xu, X. (2012). From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 28(1):75–86.

[Zeeb et al., 2007] Zeeb, E., Bobek, A., Bohn, H., and Golatowski, F. (2007). Service-oriented architectures for embedded systems using devices profile for web services. In *21st International Conference on Advanced Information Networking and Applications Workshops, 2007, AINAW '07*, volume 1, pages 956–963.

6

Appendix 1 - WSDL file for the Crane Service

Listing 6.1: Inico S1000 programming environment - example of ST program

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="Crane" targetNamespace="http://www.uninova.pt/wsdl/Crane"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl="http://schemas.xmlsoap.
org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.uninova.pt/wsdl/Crane"
xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
 <types>
  <xsd:schema targetNamespace="http://www.uninova.pt/wsdl/Crane"
  elementFormDefault="qualified">

    <xsd:element name="pickRequest">
    </xsd:element>

    <xsd:element name="pickResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="pickOutput" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="placeRequest">
    </xsd:element>

    <xsd:element name="placeResponse">
```

```
28        <xsd:complexType>
29          <xsd:sequence>
30            <xsd:element name="placeOutput" type="xsd:string"/>
31          </xsd:sequence>
32        </xsd:complexType>
33      </xsd:element>
34
35      <xsd:element name="moveYRequest">
36      <xsd:complexType>
37          <xsd:sequence>
38            <xsd:element name="moveYInput" type="xsd:string"/>
39          </xsd:sequence>
40        </xsd:complexType>
41      </xsd:element>
42
43      <xsd:element name="moveYResponse">
44        <xsd:complexType>
45          <xsd:sequence>
46            <xsd:element name="moveYOutput" type="xsd:string"/>
47          </xsd:sequence>
48        </xsd:complexType>
49      </xsd:element>
50
51      <xsd:element name="moveXRequest">
52      <xsd:complexType>
53          <xsd:sequence>
54            <xsd:element name="moveXInput" type="xsd:string"/>
55          </xsd:sequence>
56        </xsd:complexType>
57      </xsd:element>
58
59      <xsd:element name="moveXResponse">
60        <xsd:complexType>
61          <xsd:sequence>
62            <xsd:element name="moveXOutput" type="xsd:string"/>
63          </xsd:sequence>
64        </xsd:complexType>
65      </xsd:element>
66
67      <xsd:element name="moveToRefRequest">
68      </xsd:element>
69
70      <xsd:element name="moveToRefResponse">
71        <xsd:complexType>
72          <xsd:sequence>
73            <xsd:element name="moveToRefOutput" type="xsd:string"/>
74          </xsd:sequence>
75        </xsd:complexType>
76      </xsd:element>
77
```

```
78    </xsd:schema>
79  </types>
80
81  <message name="pickRequestMsg">
82    <part name="body" element="tns:pickRequest"/>
83  </message>
84  <message name="pickResponseMsg">
85    <part name="body" element="tns:pickResponse"/>
86  </message>
87  <message name="placeRequestMsg">
88    <part name="body" element="tns:placeRequest"/>
89  </message>
90  <message name="placeResponseMsg">
91    <part name="body" element="tns:placeResponse"/>
92  </message>
93  <message name="moveYRequestMsg">
94    <part name="body" element="tns:moveYRequest"/>
95  </message>
96  <message name="moveYResponseMsg">
97    <part name="body" element="tns:moveYResponse"/>
98  </message>
99  <message name="moveXRequestMsg">
100   <part name="body" element="tns:moveXRequest"/>
101 </message>
102 <message name="moveXResponseMsg">
103   <part name="body" element="tns:moveXResponse"/>
104 </message>
105 <message name="moveToRefRequestMsg">
106   <part name="body" element="tns:moveToRefRequest"/>
107 </message>
108 <message name="moveToRefResponseMsg">
109   <part name="body" element="tns:moveToRefResponse"/>
110 </message>
111
112 <portType name="CraneServicePortType" wse:EventSource="true">
113
114   <operation name="pick">
115     <input message="tns:pickRequestMsg"/>
116     <output message="tns:pickResponseMsg"/>
117   </operation>
118
119   <operation name="place">
120     <input message="tns:placeRequestMsg"/>
121     <output message="tns:placeResponseMsg"/>
122   </operation>
123
124   <operation name="moveY">
125     <input message="tns:moveYRequestMsg"/>
126     <output message="tns:moveYResponseMsg"/>
127   </operation>
```

```
128
129   <operation name="moveX">
130     <input message="tns:moveXRequestMsg"/>
131     <output message="tns:moveXResponseMsg"/>
132   </operation>
133
134   <operation name="moveToRef">
135     <input message="tns:moveToRefRequestMsg"/>
136     <output message="tns:moveToRefResponseMsg"/>
137   </operation>
138
139 </portType>
140
141 <binding name="CraneServiceBinding" type="tns:CraneServicePortType">
142
143     <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
144     style="document" />
145
146   <operation name="pick">
147     <soap:operation style="document" />
148     <wsdl:output>
149       <soap:body use="literal" />
150     </wsdl:output>
151   </operation>
152
153   <operation name="place">
154     <soap:operation style="document" />
155     <wsdl:output>
156       <soap:body use="literal" />
157     </wsdl:output>
158   </operation>
159
160   <operation name="moveY">
161     <soap:operation style="document" />
162     <wsdl:input>
163       <soap:body use="literal" />
164     </wsdl:input>
165     <wsdl:output>
166       <soap:body use="literal" />
167     </wsdl:output>
168   </operation>
169
170   <operation name="moveX">
171     <soap:operation style="document" />
172     <wsdl:input>
173       <soap:body use="literal" />
174     </wsdl:input>
175     <wsdl:output>
176       <soap:body use="literal" />
177     </wsdl:output>
```

```
178   </operation>
179
180   <operation name="moveToRef">
181     <soap:operation style="document" />
182     <wsdl:output>
183       <soap:body use="literal" />
184     </wsdl:output>
185   </operation>
186
187 </binding>
188
189 <service name="Crane">
190   <port name="CraneServicePort" binding="tns:CraneServiceBinding">
191     <soap:address location="http://192.168.3.14:80/dpws/Crane" />
192   </port>
193 </service>
194
195  </definitions>
```