

Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit

Evaluation and Integration of an OPC UA Service into a Container-Based Cloud Platform

Viktor Krimstein

Course of Study: Softwaretechnik

Examiner: Prof. Dr. Stefan Wagner

Supervisor: Carsten Ellwein, M.Sc.
Dipl.-Ing. Wolfgang Fechner

Commenced: November 22, 2017

Completed: May 22, 2018

Abstract

In the context of Industry 4.0, services are brought to the forefront along with a higher degree of networking. The vision of self-governing systems that exchange information and make decisions without human intervention is a primary goal but not necessarily the focus of many projects.

The research project “Rent’n’Produce: Secure Cloud Service for the Commissioning and Control of Production Systems” which is currently being processed at the Institute for Control Engineering of Machine Tools and Manufacturing Units of the University of Stuttgart, has the vision of advancing self-directed manufacturing systems. Companies from different industries should be provided with a Cloud-based platform that enables a highly flexible production order.

This bachelor thesis focuses on the integration of a real machine tool into the Rent’n’Produce Cloud Platform by using the Open Platform Communications Unified Architecture. The required functionality of the integrated service should be encapsulated in a virtualization container. This middleware is intended to provide access to configuration and status data of the production resource, to transfer numerical control programs to the machine tool as well as to start or stop the part production.

Kurzfassung

In heutigen Unternehmen werden zunehmend Konzepte und Technologien des Cloud Computings implementiert. Hierbei können Speicher- und Rechenkapazitäten von Drittanbietern in die eigenen Unternehmensprozesse eingebunden und nach Belieben skaliert werden. Im Rahmen der Industrie 4.0 finden Cloud Computing-Paradigmen zunehmend auch im Automatisierungs- und Steuerungsumfeld ihre Verwendung. Steuerungssysteme, die ganz ohne menschliches Zutun Produktionsprozesse abbilden und durchführen können, stehen jedoch nicht im Fokus aktueller Forschungsprojekte.

Das Forschungsprojekt „Rent’n’Produce: Sicherer Cloudservice zur Beauftragung und Steuerung von Fertigungssystemen“, welches aktuell am Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart bearbeitet wird, will genau diese Lücke schließen. Konsumenten und Produzenten unterschiedlicher Wirtschaftszweige soll eine cloud-basierte Plattform zu Verfügung gestellt werden, mit der Automatisierungsprozesse bis hin zu administrativen Operationen auf Produktionsressourcen realisiert werden können.

Ziel dieser Arbeit ist die Anbindung einer realen Werkzeugmaschine basierend auf der Open Platform Communications Unified Architecture an Rent’n’Produce umzusetzen. Die benötigte Funktionalität der Anbindung soll dabei in einem Virtualisierungscontainer gekapselt und über eine standardisierte Webschnittstelle zur Verfügung gestellt werden. Diese Middleware soll den Zugriff auf Konfigurations- und Zustandsdaten der Produktionsressource, die Übertragung von Programmen für numerische Steuerungen auf die Maschine sowie das Starten sowie Stoppen von Fertigungsschritten ermöglichen.

Contents

1	Introduction	13
1.1	Motivation	14
1.2	Goals	14
1.3	Structure	15
2	Foundations	17
2.1	Rent'n'Produce	17
2.1.1	Initial Architecture	18
2.2	Cloud Computing	19
2.3	Cloud Manufacturing	20
2.4	Open Platform Communications Unified Architecture	22
2.4.1	Server Model	23
2.4.2	Client Model	24
2.5	Conclusion	25
3	State of the Art	27
3.1	Spring Boot	27
3.2	Container-Based Virtualization	27
3.3	Representational State Transfer	28
3.4	Conclusion	29
4	State of Research	31
4.1	State of Research in Cloud Manufacturing	31
4.2	Control Engineering in the Cloud	32
4.3	Machine-to-Machine Communication	32
4.4	Conclusion	33
5	Evaluation	35
5.1	Requirements	35
5.2	Decision	36
5.3	Conclusion	39
6	Concept	41
6.1	Requirements	41
6.1.1	Infrastructure	41
6.1.2	Architecture	41
6.1.3	Technology	42
6.1.4	Mandatory Features	42
6.1.5	Optional Use Cases	48

6.2	Approach	48
6.2.1	Integration	48
6.2.2	Target Architecture	49
6.2.3	Services	50
6.2.4	Workflow Concept	51
6.2.5	Machine Tool Integration	54
6.2.6	Security	55
6.3	Conclusion	56
7	Discussion	57
7.1	Main Objective	57
7.2	Approach	58
7.3	Integration Results	58
7.4	Use Case Implementation	59
7.4.1	Uploading of Production Data	59
7.4.2	Machine-Tool Status Checking	59
7.4.3	Production Data Transfer	60
7.4.4	Part Production	61
7.5	Limitations	61
7.6	Conclusion	62
8	Conclusion and Future Work	63
8.1	Conclusion	63
8.2	Future Work	63

List of Figures

2.1	The workflow of the Rent'n'Produce platform adapted from [Ell16]	18
2.2	Simplified architecture of the Rent'n'Produce platform	19
2.3	Architecture of a Cloud Manufacturing system adapted from [Xu12]	22
2.4	The Open Platform Communications Unified Architecture server architecture adapted from [OPC17a]	24
2.5	The Open Platform Communications Unified Architecture client architecture adapted from [OPC17a]	25
3.1	Diagrams showing hypervisor-based and Docker virtualization	28
6.1	The Use Case diagram for the Proof-of-Concept	42
6.2	The component diagram of the integration of the Proof-of-Concept	49
6.3	The component diagram describing service details of the Proof-of-Concept	50
6.4	Sequence diagram of the general part production process	53

List of Tables

3.1	Hypertext Transfer Protocol to Representational State Transfer Mapping	29
5.1	Comparison of Open Platform Communications Unified Architecture Frameworks	38
6.1	Use Case description for the upload of production data	44
6.2	Use Case description for the retrieval of the current machine tool status	45
6.3	Use Case description for transfer of production data to the machine tool	46
6.4	Use Case description for the production control of the machine tool	47

List of Abbreviations

- AMQP** Advanced Message Queuing Protocol. 51
- API** Application Programming Interface. 15
- C#** The Microsoft Visual C-Sharp Programming Language. 35
- C/C++** The C11 and C++17 Programming Languages. 35
- CAD** Computer-Aided Design. 14
- CAM** Computer-Aided Manufacturing. 14
- CM** Cloud Manufacturing. 13
- COM/DCOM** Component Object Model/Distributed Component Object Model. 22
- CPS** Cyberphysical System. 13
- CPU** Central Processing Unit. 27
- CRM** Customer Relationship Management. 13
- CRUD** Create, Read, Update, Delete. 28
- DAMA** Design Anywhere, Manufacture Anywhere. 13
- DDS** Data Distribution Service. 13
- ERP** Enterprise Resource Planning. 13
- G-Code** International Organization for Standardization 6983 Code. 14
- GoLang** The Google Go Programming Language. 35
- HMI** Human Machine Interface. 48
- HTTP** Hypertext Transfer Protocol. 28
- IaaS** Infrastructure as a Service. 19
- IoT** Internet of Things. 13
- ISW** Institute for Control Engineering of Machine Tools and Manufacturing Units of the University of Stuttgart. 13
- IT** Information Technology. 23
- Java** The Oracle Java Programming Language. 27
- JPA** Java Persistence Application Programming Interface. 50

- JSON** JavaScript Object Notation. 27
- LXC** Linux Containers. 27
- M2M** Machine-to-Machine. 13
- MRP** Manufacturing Resource Planning. 13
- NC** Numerical Control. 14
- NIST** National Institute of Standards and Technology of the United States of America. 19
- NodeJS** The NodeJS Javascript Programming Language. 35
- NoSQL** Not only SQL. 18
- OPC** Open Platform Communications. 13
- OPC UA** Open Platform Communications Unified Architecture. 13
- OS** Operating System. 27
- PaaS** Platform as a Service. 19
- PLC** Programmable Logic Controller. 21
- PoC** Proof-of-Concept. 14
- Python** The CPython Programming Language. 35
- QoS** Quality of Service. 13
- R'n'P** Rent'n'Produce: Secure Cloud Service for the Commissioning and Control of Production Systems. 13
- RDBMS** Relational Database Management System. 48
- REST** Representational State Transfer. 27
- SaaS** Software as a Service. 19
- SDK** Software Development Kit. 37
- SOA** Service-Oriented Architecture. 18
- SQL** Structured Query Language. 18
- UI** User Interface. 19
- UML** Unified Modeling Language. 17
- URL** Uniform Resource Locator. 19
- VPN** Virtual Private Network. 31
- XaaS** Everything is treated as a Service. 20
- XML** Extensible Markup Language. 27

1 Introduction

The trend of connecting physical objects with the Internet gave birth to the term Internet of Things (IoT). IoT envisions applications running with interconnected objects working together to gather data and act in environments by controlling different actuators to enable new functionality with minimal human intervention [AIM10]. With the introduction of IoT and Cyberphysical System (CPS) concepts in industrial application scenarios, automation is undergoing a tremendous change [WSJ17]. By adapting these concepts, the philosophy of “Design Anywhere, Manufacture Anywhere (DAMA)” has emerged in the past decade [Hei05; Man11; VOX+05]. The DAMA approach demands the ability to exchange design and manufacturing data across multiple sites. DAMA also helps establish links between Manufacturing Resource Planning (MRP), Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) [Xu12].

However, today’s networked manufacturing mainly refers to integration of distributed resources for undertaking a single manufacturing task [LZW+10]. What is lacking in this type of manufacturing regime are the centralized operation management of the services, choice of different operation modes and embedded access of manufacturing equipment and resources, without which a seamless, stable and high quality transaction of manufacturing resource services and a Quality of Service (QoS) cannot be guaranteed [THZ10]. Further, any attempt to steer processes independently of continuous human interaction requires, in a very wide sense, the flow of information between some kind of sensors, controllers and actuators [SSKD11].

In the context of Cloud Manufacturing (CM), services with a higher degree of networking are brought to the fore. Approaches in this regard often focus on the collection and analysis of data. The vision of self-steering systems that share information and make decisions without human intervention is a primary goal, but not necessarily the focus of many projects. With the research project “Rent’n’Produce: Secure Cloud Service for the Commissioning and Control of Production Systems (R’n’P)” [Eli16] which is currently being processed at the Institute for Control Engineering of Machine Tools and Manufacturing Units of the University of Stuttgart (ISW), and its available functionalities at the time of writing this thesis, manufacturers of different industries and sectors can use the Cloud platform for flexible production assignment and detailed order management. Furthermore, one goal of R’n’P is to integrate features and functionalities to access machine tools directly from the Cloud and independently of their physical location and the underlying manufacturing platforms by relying only on a homogeneous Machine-to-Machine (M2M) communication standard.

Unfortunately, the last requirement is quite a difficult one as manufacturers show a huge heterogeneity in the ways and protocols, the machine tools support for communication and data access [Xu12]. Tackling this challenge two specifications emerged within the last years. European manufacturers integrated the Open Platform Communications Unified Architecture (OPC UA) is the extension of its predecessor Open Platform Communications (OPC), while Data Distribution Service (DDS) gets more popular within the United States [NBK+15].

1.1 Motivation

CM is a concept providing a model to transition traditional manufacturing processes into the Cloud while enabling users of CM platforms a cost-effective and efficient way to manage manufacturing resources, production workflows and control units [Xu12]. By adapting principles of Cloud Computing, manufacturers are given the possibility to track, manage and control their machine tools and manufacturing units remotely [WGRS13]. It is possible to follow the whole manufacturing process through the whole automation pyramid allowing manufacturers to scale and schedule their production planning and execution according to customers' needs [Kle14].

Currently the most Use Cases of existing CM platforms focus on the retrieval and gathering of data from machine tools. Remote control of the machines and manufacturing processes beyond company boundaries is not in the focus of research. The motivation of this work is to provide an architectural approach to provision and manage machine tools of manufacturers remotely and to transfer production data down to the machines using a standard M2M communication protocol. This work wants to provide a manufacturer and control unit independent approach that allows manufacturers to control their production processes and not only focus on monitoring them. Therefore, a real machine tool should be integrated into the R'n'P platform using the Proof-of-Concept (PoC).

As the control unit of the machine tool, a Beckhoff TwinCAT V3¹ will be used, as it implements nearly the whole OPC UA specification, according to [Bec18]. By only relying on the OPC UA protocol and the functionalities implemented by the Beckhoff TwinCAT V3 control unit, this work wants to prove, that a machine tool and control unit independent implementation of a CM platform can be realized. Additionally, this work aims to provide a solid base for future work on the field of remote automation.

1.2 Goals

The goal of this thesis is to model the integration of a real machine tool into the R'n'P Cloud platform by using OPC UA. Therefore, Use Cases will be researched and evaluated to create an architecture, modeling the specified requirements. This architecture will later on be tested, using a prototypical implementation of a PoC. The required functionality of the integrated PoC should be encapsulated in a virtualization container.

This middleware is intended to provide access to configuration and status data of the production resource, to transfer Numerical Control (NC) programs in International Organization for Standardization 6983 Code (G-Code) format², generated through a Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM) process chain, to the machine tool, and to start or stop the part production.

¹<http://www.beckhoff.de/twincat3/>

²<https://www.iso.org/standard/34608.html>

The resulting PoC should simulate the CM processes beginning with the order management and ending with the production of a real part. Subgoals are defined as follows:

1. Evaluation and specification of an architecture providing the required functionality.
2. Implementing and integrating a PoC service based on the developed architecture.
3. The PoC should provide its functionality using an Application Programming Interface (API).
4. The PoC should be able to store G-Code for the production part description.
5. Integration of a real milling machine to the R'n'P platform using the PoC.

1.3 Structure

This thesis is structured as follows: In Chapter 2, the scope of this work is described in detail and a brief introduction to the related research field of this work is provided. Furthermore, relations to the R'n'P research project and to the field of CM are provided as well. References to the current state of the art are presented in Chapter 3 and will show the stack of technologies and techniques used in this work. Afterwards a differentiated view on the current state of the research is taken in Chapter 4. In this chapter the general state of current research on the topics that relate to this thesis are described by at the same time distancing from the current industrial state. Chapter 5 then describes the evaluation of a suitable Open Source OPC UA framework for the implementation of the PoC. Concepts and methodologies of the PoC to be developed as part this work are discussed in more detail in Chapter 6. In Chapter 7 the results of the PoC implementation are discussed. Chapter 8 concludes the paper and outlines possible future work.

2 Foundations

In Chapter 1 a short summary on CM and especially of the challenges in terms of M2M communication and the current topology at manufacturing sites was given as well as a motivation, why the integration of new features is beneficial for the research state of R'n'P. To better understand the used terms and suggested concepts, this chapter introduces fundamental knowledge related to this work.

A closer description of the research project R'n'P is provided in Section 2.1. The underlying concepts of Cloud Computing are highlighted in Section 2.2. Its' applications to manufacturing models are provided in Section 2.3 including the CAD-CAM process chain. Finally, Section 2.4 gives an introduction to the M2M communication protocol used in this work.

2.1 Rent'n'Produce

The research project R'n'P of the ISW draws on the existing infrastructure and networking in manufacturing companies and relies on a Cloud-based platform for the commissioning of machine tools and equipment [ER17]. The declared goal of the research project is a highly flexible, location-independent production order across company boundaries. The production parts' description required for commissioning can be present in the form of CAD data or NC programs and transmitted to the system by the client.

The provided functionality of the R'n'P platform at the beginning of this work is show in Figure 2.1. Figure 2.1 is drawn in Unified Modeling Language (UML)-like syntax for Interaction Diagrams, as described in [Obj17], to show the interactions of the different entities in the R'n'P platform. The R'n'P platform provides its' functionalities in a cyclic manner. As shown in Figure 2.1, Producers can signal the R'n'P resource management, their current production state and machine tool occupation. The R'n'P middleware then can signal to the Customers, that potential manufacturing resources are available from the Producer or even multiple Producers. In top-down order, Figure 2.1 shows that Customers can assign production orders to one or multiple Producers, using the Resource Assignment Layer of the R'n'P platform.

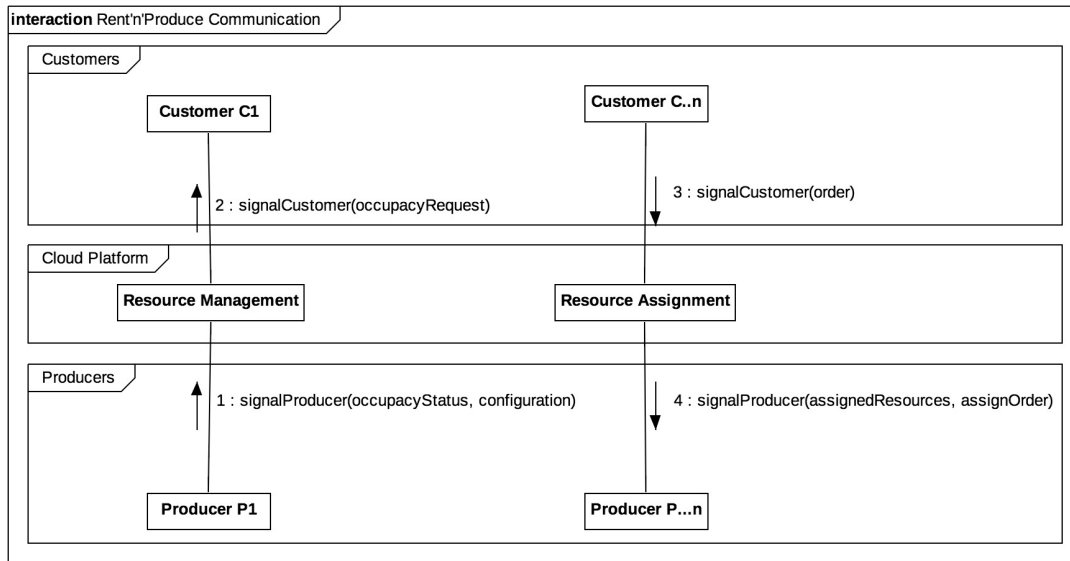


Figure 2.1: The workflow of the Rent'n'Produce platform adapted from [EII16]

Before commissioning the selected resource at the shop-floor level (the production level at the producer's manufacturing site), the production part descriptions have to go through a postprocessor and then to be adapted to the selected production resource from the resource catalog. This thesis deals with the transfer of existing G-Code data to a production resource. The postprocessing of workpiece descriptions is part of a parallel research project.

2.1.1 Initial Architecture

The architecture is based on several services which work independently of each other following a Service-Oriented Architecture (SOA) approach [Erl08]. Each service is split up in a persistence layer and a business logic layer making its functionality accessible to other services using an API. A simplified view on the architecture of R'n'P is shown in Figure 2.2 described as UML Component Diagram following [Obj17].

The persistence layer, shown in Figure 2.2, contains the databases (Structured Query Language (SQL) or Not only SQL (NoSQL)) for the different services in the overall SOA. These databases offer their APIs to the Service Layer, implementing the business logic of the whole R'n'P platform. As presented in Figure 2.2, the service layer is encapsulated by the API Gateway Component, bridging the APIs of the R'n'P platform to the Web Interface, shown to the end-users of R'n'P.

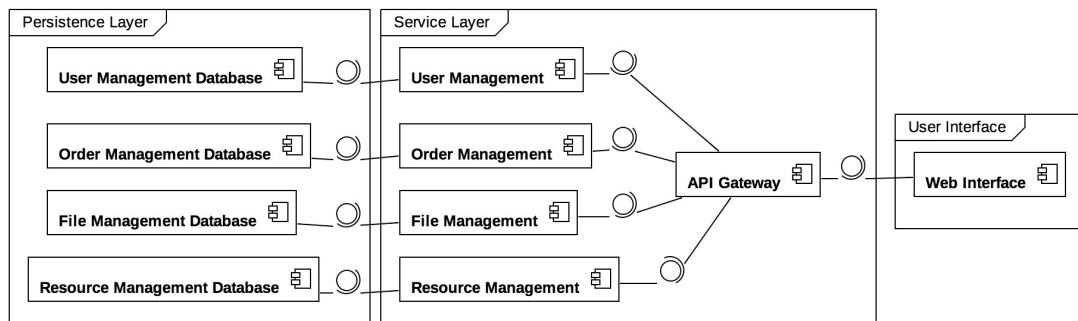


Figure 2.2: Simplified architecture of the Rent'n'Produce platform

Further, every service is responsible only for one specific part of the whole platform domain (e.g., order management, user access and permission management etc.). Through the server database interface, the business logic can access the persistence layer to store and retrieve data required for domain specific operations. The business logic layer itself, contains all functionality provided by the platform encapsulated to different services.

Services and databases are encapsulated in virtualization containers to ensure platform independence and the increase of maintainability [ER17]. The web interface component is the application layer which contains the User Interface (UI) and the API gateway routing all client requests to the appropriate services by its internal Uniform Resource Locator (URL) representation.

2.2 Cloud Computing

By the National Institute of Standards and Technology of the United States of America (NIST) definition of Cloud Computing, Cloud Computing is 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 [MG+11]. This cloud model is composed of five essential characteristics, three service models, and four deployment models [FLR+14]. With regard to this work, the following paragraphs will focus and describe the three service models, often adapted in the manufacturing environment: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Software as a Service The capability provided to the consumer is to use the provider's applications running on a Cloud infrastructure [MG+11]. A Cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of Cloud Computing [MG+11]. The Cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer [FLR+14]. The physical layer consists of the hardware resources that are necessary to support the Cloud services being provided, and typically includes server, storage and network components [FLR+14]. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential Cloud characteristics [FLR+14]. Conceptually the abstraction layer sits above the physical layer [FLR+14].

The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface [MG+11]. The consumer does not manage or control the underlying Cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited userspecific application configuration settings [MG+11].

Platform as a Service The capability provided to the consumer is to deploy onto the Cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider [FLR+14]. This capability does not necessarily preclude the use of compatible programming languages, libraries, services, and tools from other sources [Ley11].

The consumer does not manage or control the underlying Cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment [Ley11].

Infrastructure as a Service The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications [Ley11]. The consumer does not manage or control the underlying Cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls) [MG+11].

As shown in Section 2.1, the R'n'P Cloud platform is built upon a IaaS layer providing its functionality over the PaaS and SaaS models [ER17]. The goals and motivations of this work, can be related to the PaaS model, as machin-tools and manufacturing sites should be provisioned over the services provided by the platform.

2.3 Cloud Manufacturing

With the broad adoption of Cloud Computing concepts and technology in the software industry, especially the use of virtualization, shared resources, outsourcing of high performance workloads, SOA and the concepts described in Section 2.2, CM emerged and is getting adapted more and more in the production environment [HX15]. CM models the relocation of production resources via the internet either at times of equipment peaks or from cost reasons [WGRS13]. Further, CM allows its users to access a pool of manufacturing services and resources in a flexible and location independent (only limited by the real-time connectivity required) manner, using an Everything is treated as a Service (XaaS) payment model only taking the usage time into account [MBM+12].

Services offered to the manufactures range from product design and development to manufacturing production of the requested parts [Xu12]. The provided resources are managed and encapsulated in a centralized way through the whole automation pyramid [Kle14]. In a CM system, three types of users and roles are defined – providers, consumers and operators [WGRS13].

Providers offer manufacturing resources on the platform. Their roles and respective representations can vary from private customers, small business to specialized manufacturing service

providers [TCD+14]. Clients occupy and use the manufacturing resources provided by the platform by paying for the usage time of the specific machine tools [HX15]. Last but not least operators are responsible for operating and maintaining the services provided on the platform as well as to deliver services which can be used by both parties mentioned above [Xu12]. Moreover, operators are hold accountable to maintain and modernize the service stack within the platform including, architecture, technologies and security related methodologies [TMTR15].

The architecture of a CM platform shown in Figure 2.3 in a UML-like Interaction Diagram following [Obj17] is constructed by the following four tiers and three domains: the manufacturing resource layer, virtual service layer, global service layer and the application layer [WGRS13]. The product life cycle of manufacturing, including machine tools, gateways and administration tools, is represented by the provider domain and offered by the manufacturing resource layer. Resources can either be any kind of CPS as well as generalized and abstracted manufacturing capabilities [Kle14]. Manufacturing resources include for instance equipment, servers or Programmable Logic Controller (PLC) units [BFKR14].

Abstract manufacturing capabilities describe the ability of producers and manufacturer to offer the possibility for executing specialized, non typical tasks including the associated design, planning and management processes culminating in production site specific hardware and administration tools [HX15]. The virtual service layer concludes the provider domain by abstracting the underlying manufacturing resource layer and acting as an interface for the global service layer [Xu12].

Cloud deployment technologies necessary to run the whole CM system and to bridge the provider and user domains are implemented by the global service layer. By mentioning Cloud deployment technologies this work refers to IaaS or a PaaS offering, as described in Section 2.2. Two operational modes are enabled for the global service layer – complete service mode and partial service mode. The complete service mode orchestrates all manufacturing processes itself [Xu12]. In the partial service mode, it is possible for providers to take control over the processes at system scope where the global service layer supports the management and organization of these processes.

Finally, the application layer rounds off the CM model by being responsible for end-user demand and manufacturing process management mapping [Xu12].

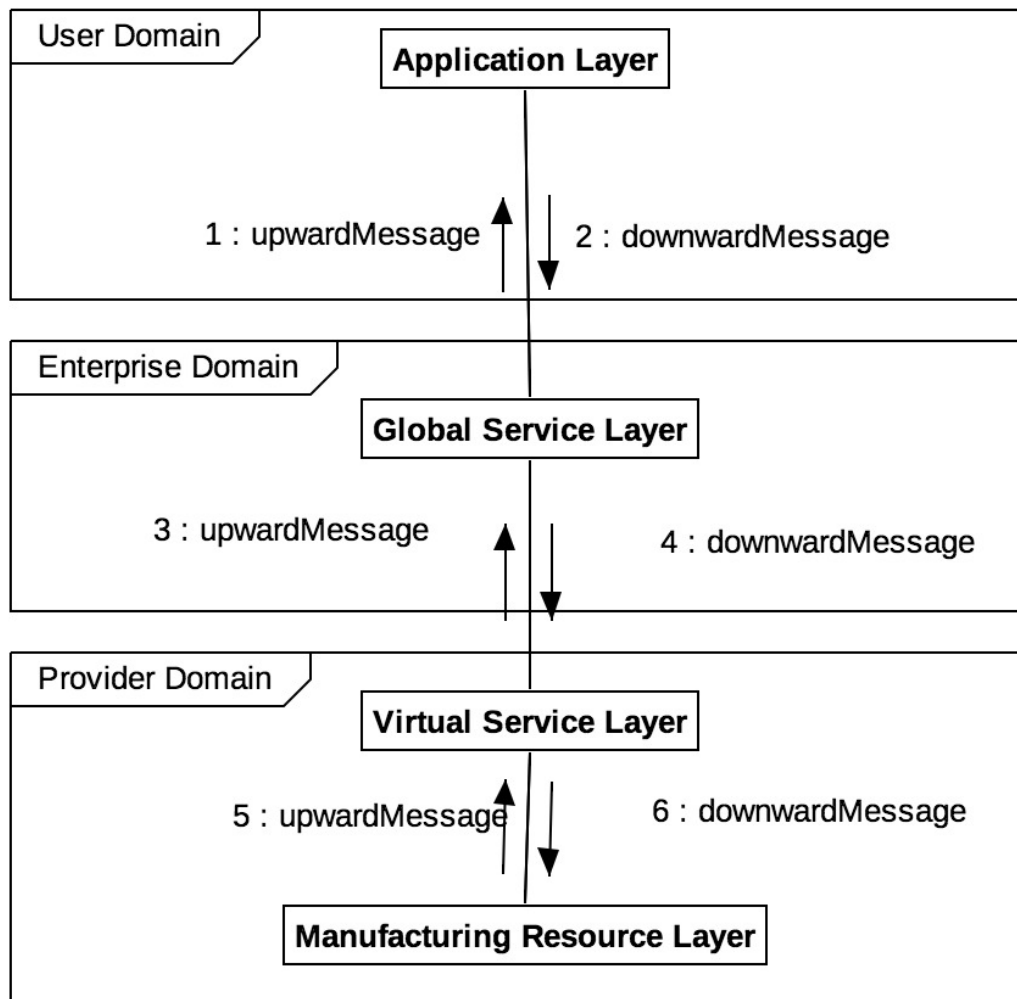


Figure 2.3: Architecture of a Cloud Manufacturing system adapted from [Xu12]

2.4 Open Platform Communications Unified Architecture

The OPC UA is the new version of the well-known OPC architecture [Had06] originally designed by the OPC Foundation to connect industrial devices to control and supervision applications [HS14]. The focus of OPC is on getting access to large amounts of real-time data while ensuring performance constraints without disrupting the normal operation of the devices [CJOC10]. The original OPC specifications, based on Microsofts Component Object Model/Distributed Component Object Model (COM/DCOM), are becoming obsolete and are gradually being replaced by new interoperability standards, including web-services what led the OPC Foundation to publish a new architecture, called OPC UA [Had06].

Notice The following two figures, shown in Figure 2.4 and Figure 2.5 were cited with the kind permission of the OPC Foundation to cite this images by referencing the origin OPC UA Specification Part referenced in [OPC17a].

2.4.1 Server Model

OPC UA specifies the exchange of real-time information of production plant data between control devices or Information Technology (IT) systems from different manufacturers [VOX+05]. The communication is established by an *inverted* client-server system, where the client triggers actions on the server for automation control, and the server executes the commands on, or retrieves data from the underlying machine [IJ13].

Figure 2.4 shows the OPC UA Server Model according to its specification in [OPC17a]. OPC UA servers include an information model that allows users to organize data and their semantics in a structured manner.

This semantic *AddressSpace* is constructed of standalone or interconnected *Nodes* mapped to real CPS object representatives as shown in Figure 2.4. Furthermore, nodes can be divided into functionality and view nodes, each sort implementing different functionalities and manners of user interaction. Further, each node can be monitored and subscribed by parties of interest using the OPC UA Server API as presented at the bottom of Figure 2.4. The information model constitutes the address spaces of OPC UA servers. It is a fullmesh network of nodes with their properties and relations.

In general, users create the information model for their OPC UA servers manually at implementation time or implement vendor-specific automatism [HS14]. A server address space consists of the following element types which are shown in Figure 2.4.

- *Object*: “A *Node* that represents a physical or abstract element of a system. Objects are modeled using the OPC UA *Object Model*. Systems, subsystems and devices are examples of Objects. An Object may be defined as an instance of an *ObjectType*.” [OPC17a]
- *ObjectType*: “A *Node* that represents the type definition for an Object.” [OPC17a]
- *Variable*: “A *Variable* is a *Node* that contains a value.” [OPC17a]
- *VariableType*: “*Node* that represents the type definition for a *Variable*” [OPC18]
- *DataType*: “An instance of a *DataType Node* that is used together with the *ValueRank* attribute to define the data type of a *Variable*.” [OPC18]
- *ReferenceType*: “A *Node* that represents the type definition of a *Reference*. The *ReferenceType* specifies the semantics of a *Reference*. The name of a *ReferenceType* identifies how source *Nodes* are related to target *Nodes* and generally reflects an operation between the two, such as “A Contains B”.” [OPC17a]
- *Method*: “A callable software function that is a component of an Object.” [OPC17a]
- *View*: “A specific subset of the *AddressSpace* that is of interest to the *Client*.” [OPC17a]

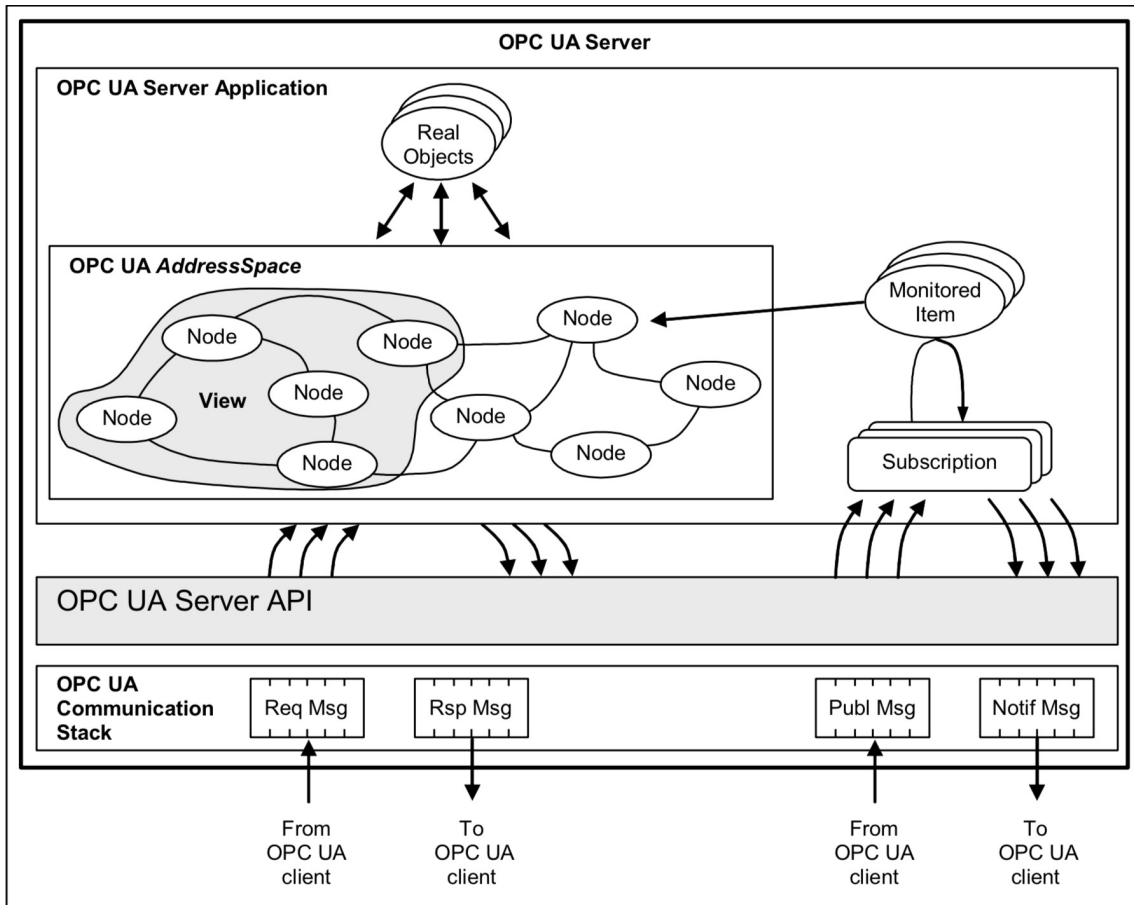


Figure 2.4: The Open Platform Communications Unified Architecture server architecture adapted from [OPC17a]

2.4.2 Client Model

The OPC UA Client architecture models the Client endpoint of client/server interactions. Figure 2.5 illustrates the major elements of a typical Client and how they relate to each other. As presented in Figure 2.5, the OPC UA Client is constructed by two layers – the Client Application and the OPC UA Client API. The Client Application encapsulates the producer–consumer service functionality by accessing the underlying OPC UA Client-API following the asynchronous system designs described in [TV07].

Further, the Client Application is the code that implements the function of the Client. It uses the Client API to send and receive OPC UA Service requests and responses to the Server. The Services defined for OPC UA are described in Clause 6.4 of [OPC17b]. Note that the “Client API” is an internal interface that isolates the Client application code from an OPC UA Communication Stack.

The OPC UA Communication Stack converts Client API calls into Messages and sends them through the underlying communications entity to the Server at the request of the Client application. The OPC UA Communication Stack also receives response and *NotificationMessages* from the underlying communications entity and delivers them to the Client application through the Client API.

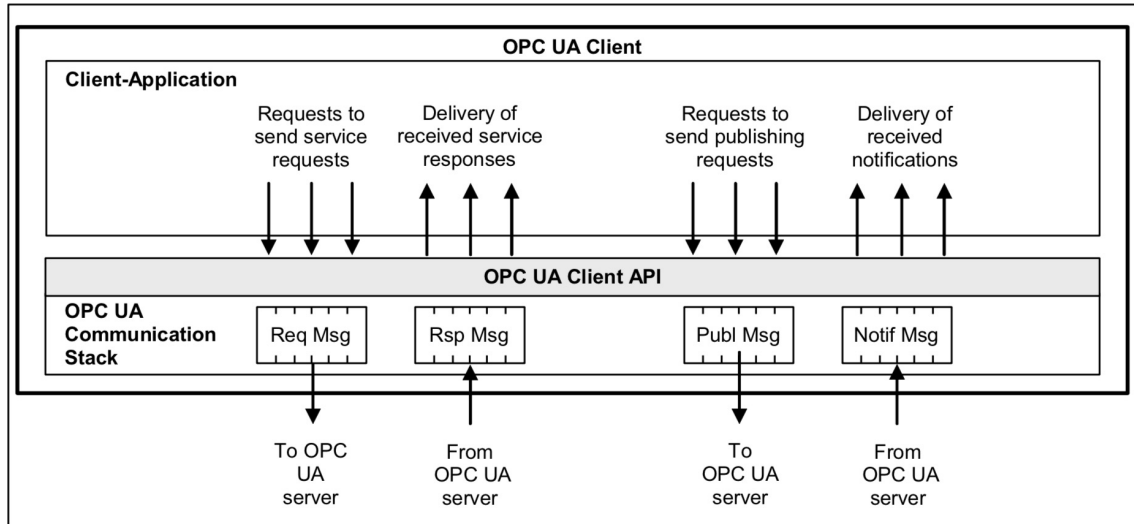


Figure 2.5: The Open Platform Communications Unified Architecture client architecture adapted from [OPC17a]

2.5 Conclusion

In this section a detailed introduction to the R'n'P CM platform has been provided. The concepts of Cloud Computing and CM used in the platform have been provided as well. OPC UA has been introduced as an emerging M2M communication protocol and its Server and Client APIs and architectures have been described in more detail.

3 State of the Art

In this chapter this work covers the state-of-the-art concepts and technologies being used for the evaluation, implementation and integration of the PoC. Section 3.1 introduces the Spring Boot framework for The Oracle Java Programming Language (Java) [GJS+15] programming language. In Section 3.2 the trending virtualization container technology Docker is highlighted. Finally, the Representational State Transfer (REST) concept for API design is covered by Section 3.3

3.1 Spring Boot

The SOA of the R'n'P platform was realized using the Java [GJS+15] programming language in combination with the Spring Boot Framework¹. The framework provides enterprise ready and production based patterns and functionalities for building backend services in Cloud architectures. Spring Boot provides a common abstraction layer of database entities, repositories abstracting the underlying persistence layer, services implementing the business logic and REST controller presenting the business logic as REST-ful API including the mapping of different data models like JavaScript Object Notation (JSON) or Extensible Markup Language (XML) to Java [GJS+15] objects.

Besides its clean architectural approach, Spring Boot provides standard enterprise pattern configurations for message broker, service discovery, performance and metrics tracing as well as custom web-server configurations with little to no configuration effort.

3.2 Container-Based Virtualization

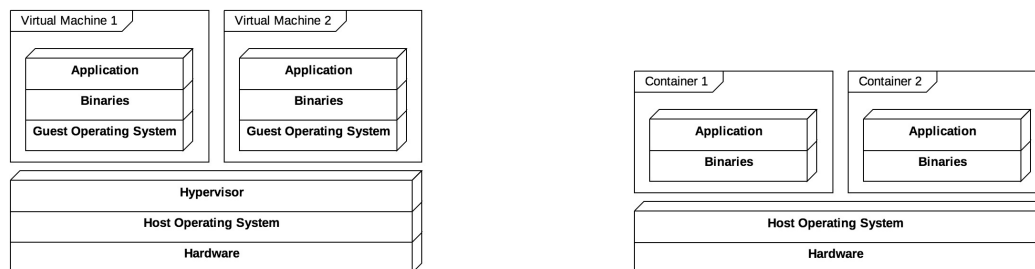
Docker² is an open source project providing a systematic way to automate the faster deployment of Linux applications inside portable containers [Ber14]. Basically, Docker extends Linux Containers (LXC) with a kernel-and application-level API that together run processes in isolation: Central Processing Unit (CPU), memory, disk read and write operations, network, and so on. Docker also uses namespaces to completely isolate an application's view of the underlying operating environment, including process trees, network, user identifiers, and file systems. Docker containers are created using base images. A Docker image can include just the Operating System (OS) fundamentals, or it can consist of a sophisticated prebuilt application stack ready for launch.

¹<https://projects.spring.io/spring-boot/>

²<https://www.docker.com/>

When building images with Docker, each action taken (that is, command executed, such as an installation of dependencies) forms a new layer on top of the previous one. Commands can be executed manually or automatically using Dockerfiles.

The difference between standard hypervisor virtualization and Docker containers is shown in Figure 3.1. Each block, shown in Figure 3.1a and Figure 3.1b describes a physical or software layer, of the virtualization process. By referencing the hardware layer, this work refers to the underlying physical device either represented by a server or general computer. The Host Operating system refers to the Operating System installed upon the underlying hardware architecture. In hypervisor-based virtualization, the hypervisor is a software product, installed upon the underlying hardware and host operating system, providing hardware, network and process protocol virtualization functionalities. The blocks in the first virtual machine described in Figure 3.1a represents a virtualized computer, containing its own Operating System, binaries containing encapsulated system functionalities to run specific software of the Operating System and the Application as the piece of software to be run in the virtual machine. Analogously, these terms and symbols find their use in Figure 3.1b.



(a) A diagram showing the layers of a hypervisor-based virtualization

(b) A diagram showing the layers of a Docker container virtualization

Figure 3.1: Diagrams showing hypervisor-based and Docker virtualization

While standard virtualization builds each operating system itself, container rely only on the presence of the Docker Engine, encapsulating services in minimal environments. Docker Compose³ is a tool for defining and running multi-container Docker applications. It offers configuration, management and orchestration utilities for container based or hybrid architectures. Further, Docker Compose can be used to deploy Docker containers and its services to different hosting environments by guaranteeing the same deployment and service architecture on each of the different environments.

3.3 Representational State Transfer

REST is a pattern of resource operations that has emerged as a *de facto* standard for service design in Web 2.0 applications [BB08]. Whereas the traditional approaches to Web Services uses full-blown remote objects with remote method invocation and encapsulated functionality, REST deals only with data structures and the transfer of their state [FT00].

³<https://docs.docker.com/compose/>

At the core of REST based design is a set of state transfer operations universal to any data storage and retrieval system. These operations, as commonly interpreted on the web, are referred to by the acronym Create, Read, Update, Delete (CRUD) [BB08]. The Web 2.0 community has adopted an informal mapping of CRUD operations onto the commands provided by the Hypertext Transfer Protocol (HTTP) protocol: POST, GET, PUT, and DELETE, respectively. These commands identify the particular CRUD operation being requested of the resource identified by the URL endpoint. Table 3.1 shows the mapping between REST and the HTTP protocol.

Table 3.1: Hypertext Transfer Protocol to Representational State Transfer Mapping

CRUD operations	HTTP command	Input format	Output format
Create	POST	HTTP Form Encoded	Status 201 CREATED
Read	GET	None	Determined by request headers
Update	PUT	HTTP Form Encoded	Status 200 OK
Delete	DELETE	None	Status 200 OK

3.4 Conclusion

In this section State of the Art technologies and concepts have been introduced. Spring Boot and the Spring Framework have been presented as a widely adopted technology to build Java Services for SOA applications. Further, concepts and technologies to distribute applications and services over different platforms and underlying systems using container virtualization with Docker have been described as well. Finally, REST has been described as a fundamental concept of modeling service APIs. The mapping process of HTTP requests to the REST model has been described as well.

4 State of Research

This chapter highlights related research and work respective to this thesis. Section 4.1 summarizes the current state of CM. In Section 4.2 a brief overview is given over the state of research in Control Engineering from the cloud. This chapter is closed by providing information about the current research state on M2M communication protocols in Section 4.3.

4.1 State of Research in Cloud Manufacturing

As a new IT paradigm, Cloud Computing is being increasingly adopted to transform the way that IT resources are utilized and consumed [RZW+17]. Consequently, the manufacturing industry is exploring Cloud Computing in order to improve existing manufacturing structures and enterprise systems, to share and provide on-demand networked manufacturing services [RZW+17], and to better satisfy specific manufacturing enterprise needs [AWHM17]. So far researchers have proposed a number of new techniques and approaches to encapsulate various virtualized manufacturing resources and capabilities as Cloud-Based services and to implement enterprise CM service frameworks and platforms [TCC+17].

But as the use of these new paradigms gets more and more common, challenges for CM service models and production sites arise [ZZZ+18]. Networked manufacturing for example, lacks strong management mechanisms and efficient tools to coordinate large-scale distributed resources, services and operations [LYZ+17].

Further, researchers aim to decentralize the management of manufacturing resources in the cloud by providing management models to control and administrate manufacturing units through the cloud [TCC+17]. Tackling these challenges [LYZ+17] proposes a multi-centric management approach for automated manufacturing unit allocation to which, the results of this work could be bind seamlessly [AWHM17].

Besides the lack of management and control mechanisms in CM multivendor systems and heterogeneous infrastructure on production sites pose additional challenges on the CM research field [TKG+17]. The specification of standard communication protocols and data models gains a fast growing role in the automation [RZW+17] and CM field [WSJ17].

Additionally, security gets more and more relevance in CM field [TKG+17]. In a traditional on-premise application deployment model, the sensitive data of each enterprise continues to reside within the enterprise boundary and is subject to its physical, logical and personnel security and access control policies [JBM+17]. However, in the SaaS model, the enterprise data is stored outside the enterprise boundary, at the SaaS vendor end [AWHM17]. Consequently, the SaaS vendor must adopt additional security checks to ensure data security and prevent breaches due to security vulnerabilities in the application or through malicious employees [CMC+17].

4.2 Control Engineering in the Cloud

As mentioned in Section 4.1, that with the emerging of Cloud Computing patterns in the manufacturing field, control engineers as well try to move their field of automation to the Cloud [SKS+17]. As discussed in [HBM+18], an approach that implements the concept of PLC in a XaaS manner was provided by their work. The result presented in [HBM+18] is a CPS including control over PLCs, implementing concepts of Cloud Computing and provided using the Cloud Computing service models. Following the security principles presented in [TSSJ17], the communication of CPS connected to the provided control platform in [HBM+18] have been secured through a Virtual Private Network (VPN) tunnel providing the public cloud PaaS functionalities as described in Sections 2.2 and 2.3. Not taking performance and real-time computing requirements into account, this paper shows that a shift to the Cloud in the automation field can be realized [HBM+18].

Another research project which has automation and control using PLCs as a focus is the project "piCASSO" currently researched at the ISW, which began in 2016 and is still continued [KFLV16]. Its goal is the auto-provisioning CPS to manufacturing sites, which relates to the goals defined in parallel with the project of R'n'P described in Section 2.1 [ER17]. A platform based on a SOA, which coordinates a robot remotely through the Cloud, has been presented in [KFLV16]. Showing different approaches of networking in the automation field, the results several new points of interest to researchers in this field [KFLV16].

4.3 Machine-to-Machine Communication

Many CPS are sensor-rich distributed real-time embedded systems that closely interact with the physical world [ZKRC17]. In such systems, numerous entities cooperate with each other to achieve their common goals [LDZ18]. They collect data from the physical world using sensors and feed the sensor data into computing resources, which in turn make real-time decisions in cooperation by sharing data and information among participating entities [RZW+17].

In the context of CM and control engineering from the Cloud, researchers focus on the definition of standard M2M communication protocols tackling on exactly these problems [LS18]. Respectively, two protocols emerged for the manufacturing field trying to fulfill the needs of manufacturers; OPC UA and DDS [PHMW18]. While OPC UA is largely adapted by European manufacturers, DDS is more popular at manufacturers in the United States [MWB17]. OPC UA builds a whole abstraction model containing, communication and data structures [MWB17]. DDS on the other hand offers a more simple and unopinionated model provided by an object definition and abstraction model [MWB17].

As described in Section 2.4, OPC UA serves an abstraction layer for M2M communication. Using OPC UA, attempts have been taken to implement a SOA middleware for control engineering from the Cloud [MBS+12]. However, these several attempts for implementing SOA-based Automation Systems have been implemented successfully, but always with non-industrial, non-standard software like Devices Profile for Web Services [LHL+17].

Of great interest for research is the connection and integration of production sites via OPC UA, which does not require a manufacturer-specific solution [LHL+17]. These integrations should follow the OPC UA model principles and rely on the specified standard implementation, ensuring a seamless integration of different manufacturing tools and units into a SOA middleware solution [LHL+17].

4.4 Conclusion

In this section, an overview over the current state of research in fields referenced by this work has been provided. Current research projects in the CM field have been presented in Section 4.1. There, common challenges of current researches have been unveiled and were implicitly set in the context of this work. An idea for future work by relying on one of the presented researches projects has been provided as well. Subsequently, current research projects trying to combine the fields of control engineering and the concepts of CM were shown. Two different approaches of integration of control units into Cloud platforms have been shown as well. Finally, this chapter has been concluded with an overview over current research in the field of M2M communication protocols.

5 Evaluation

In this chapter, the evaluation of an appropriate OPC UA framework for the PoC implementation is provided. Section 5.1 describes the requirements elaborated that should be met by the framework. Using the defined requirements, a decision about the framework is made in Section 5.2.

5.1 Requirements

This section will discuss the elaboration process of requirements and their definition, finally upon which a decision about which OPC UA framework should be used in the PoC to implemented as part of this work. It should be mentioned that functional and non-functional requirements are not separated in this decision process.

Licensing Following the motivation and goals defined in Sections 1.1 and 1.2, this work wants to provide an architectural approach and implement a PoC based on Open Source technologies. Thus, a requirement for the OPC UA framework to be chosen is, that it should be licensed under an Open Source License like the Apache V2 License¹.

Specification Compliance First and foremost, OPC UA is a specification of an M2M communication protocol and a corresponding architecture to establish this communication. As a result OPC UA does not specify a specific programming language, neither the level of accuracy a possible specification implementation should support. This leads to the fact, that the OPC UA framework chosen in this work, should be as accurate as possible in the implementation of the specification to guarantee, that all basic OPC UA functionalities can be used and tested in the implemented PoC. Therefore, the investigated frameworks should be checked for the degree of OPC UA specification compliance.

Separation of Concerns Following the principle of the separation of concerns described in [HL95], the OPC UA framework should be split into Client and Server APIs. This should be given because this work aims to implement a machine-tool and OPC UA Server API independent PoC, so the functionalities for the Server API are not required for the PoC and would only overload the PoC.

¹<https://www.apache.org/licenses/LICENSE-2.0>

Programming Language Support As this work aims to provide a concept for an PoC, which should be adaptable to different CM platforms, the chosen framework should ideally be available for different programming languages. For the sake of this work, the focus will be set only on the following programming languages: Java², The C11 and C++17 Programming Languages (C/C++)³, The Microsoft Visual C-Sharp Programming Language (C#)⁴, The CPython Programming Language (Python)⁵, The NodeJS Javascript Programming Language (NodeJS)⁶ and the The Google Go Programming Language (GoLang)⁷. These programming languages have been chosen as they are the most common programming languages adopted for backend applications according to [JG17].

The Java programming language is further defined mandatory because all backend services of the R'n'P platform are programmed using the Java programming language. This fact is described in detail in Section 6.2.3.

Level of Abstraction The presentation of an abstract concept for the implementation of a proof-of-concept is the overarching goal of this work. Therefore, the framework should abstract the underlying OPC UA Client API functionalities into bundled logical modules. This should serve two requirements: easing of the PoC development process by the same time keeping the implemented service maintainable as well as to provide a proof, that the suggested PoC is generic enough not to relying directly on OPC UA Client API base functionalities. To analyze this more abstract requirement, the frameworks will be compared to the example specification implementation of the OPC Foundation⁸.

Industrial and Community Support The last requirement is to examine the frameworks to see if they already have industrial application in the control and automation environment. Furthermore, the Open Source frameworks should be checked for the activity of their developer community, as this software does not have to undergo regular updates under maintenance contracts and because future work must be able to continue working with the proposed methodology. To measure this requirement, general activity and commit frequency in the public versioning systems of the frameworks will be measured as well as the general response time on new issues.

5.2 Decision

Taking the requirements defined in Section 5.1, this section analyzes OPC UA frameworks and decides one to be used in the PoC implementation. In this section, the chosen frameworks will be analyzed against the requirements defined in Section 5.1. The results will be presented in Table 5.1 using Harvey Balls. A full Harvey Ball ● will show, that a requirement has totally been met. If

²<https://docs.oracle.com/javase/specs/jls/se8/html/index.html>

³<https://www.iso.org/standard/68564.html>

⁴<https://docs.microsoft.com/de-de/dotnet/csharp/>

⁵<https://docs.python.org/3/>

⁶<https://nodejs.org/dist/latest-v10.x/docs/api/>

⁷<https://golang.org/ref/spec>

⁸<https://github.com/OPCFoundation/UA-Java>

requirements have not been met in full or not at all, these results are represented by the following Harvey Balls in descending order: ●, ○, ◐ and ◑.

The following frameworks have been selected for comparison by the first 32 results of a DuckDuckGo Search⁹ result for the search query "opc ua framework java" (this search query was last checked on January 31, 2018):

1. OPC Foundation UA-Java¹⁰ - the example implementation in Java implemented by the OPC Foundation.
2. PLCcom OPC UA SDK¹¹ - a commercial Java Software Development Kit (SDK) for OPC UA Server and Client implementations supporting C# as well.
3. Ascolab UA Java SDK¹² - a commercial Java SDK for general OPC UA implementations, supporting C/C++ and C# as well.
4. opcua4j¹³ - an Open Source implementation of the OPC UA Server specification in Java.
5. Prosys OPC UA Java SDK¹⁴ - a proprietary Java SDK for OPC UA compliant OPC UA Server and Client implementations.
6. Unified Automation UA Java SDK¹⁵ - a proprietary Java SDK for OPC UA compliant OPC UA Server and Client implementations, supporting C/C++ and C# as well.
7. Eclipse Milo¹⁶ - an Open Source Java library totally compliant to the OPC UA specification separated into OPC UA Client and Server SDKs and supported by industrial partners as well as by an active community.

After the frameworks were found, each of them was checked against the requirements defined in Section 5.1. The results of the evaluation are presented in Table 5.1. In the License column, only an empty or a full Harvey Ball could be given, as the requirement is, whether the framework is Open Source or not.

Regarding the compliance to the specification a half Harvey Ball was given, if the framework implements only one of the both OPC UA APIs; either the Server API or the Client API. In the column representing the results for the Separation of Concerns requirements, the results were notated with an empty Harvey Ball if the SDK has been implemented in a monolith architecture containing both, Server and Client OPC UA APIs. If the APIs have been separated into two or more different SDKs, a full Harvey Ball was given as the requirement was totally met.

Results for the support of different programming languages were annotated with a quarter Harvey Ball if the framework is only implemented for the Java programming language and for each further supported programming language the Harvey Ball was filled a quarter step fuller. The decision

⁹<https://duckduckgo.com/>

¹⁰<https://github.com/OPCFoundation/UA-Java>

¹¹<https://www.plccom.de/produkte/opc/opc-ua-client-sdks.html>

¹²<http://www.ascolab.com/en/technology-unified-architecture/technology-implementations.html>

¹³<https://code.google.com/archive/p/opcua4j/>

¹⁴<https://code.google.com/archive/p/opcua4j/>

¹⁵<https://www.unified-automation.com/products/client-sdk/java-ua-client-sdk.html>

¹⁶<https://github.com/eclipse/milo>

concerning the level of abstraction away from the basic OPC UA API, was tested by the way, how a test client implemented with the framework, can be connected to a OPC UA Server. Method and function calls of the native OPC UA Client API have been annotated with a quarter Harvey Ball and a full Harvey Ball was given if the framework abstracts the OPC UA functionalities by providing dedicated methods for each operation and abstracting the underlying protocol and handling.

Finally, for the support of the development community and the industrial support for the framework development, a quarter Harvey Ball was given if the framework is only implemented by the company or the community reaction time is about two weeks (as seen for a not handled issue by the opcua4j framework for about 2 months). Further, a half Harvey Ball was given for an active industrial development of the framework, followed by an additionally active but not responsive open developer community and concluding to a full Harvey Ball if the framework is developed by an active Open Source community and the development is actively supported by the industry.

In Table 5.1 the requirements defined in Section 5.1 are abbreviated as follows: Licensing with *Lic*, Specification Compliance with *SC*, Separation of Concerns with *SoC*, Programming Language Support with *PLS*, Level of Abstraction with *LoA* and Industrial and Community Support with *ICS*.

Table 5.1: Comparison of Open Platform Communications Unified Architecture Frameworks

Framework	Lic	SC	SoC	PLS	LoA	ICS
OPC Foundation UA-Java	●	●	○	◐	◐	●
PLCcom OPC UA SDK	○	●	●	◐	●	●
Ascolab UA Java SDK	○	●	○	◐	●	◐
opcua4j	●	◐	○	◐	◐	◐
Prosyst OPC UA Java SDK	○	●	●	◐	●	◐
Unified Automation UA Java SDK	○	●	●	◐	●	◐
Eclipse Milo	●	●	●	◐	●	●

After filtering Table 5.1 for the requirement, that the framework must be Open Source, only the OPC Foundation, the Eclipse Milo framework and opcua4j were left. As opcua4j only implements the OPC UA Server API, just the first both passed the second column of the requirements table. As the example implementation of the OPC Foundation only relies on native OPC UA API calls, the Eclipse Milo framework has been chosen as it provides a higher layer of abstraction making the PoC developed with it more understandable and maintainable.

5.3 Conclusion

In this chapter requirements for the decision process of an OPC UA framework to be used for the implementation of the PoC in this work, have been defined. These requirements have been checked against beforehand researched OPC UA frameworks and results have been presented in Table 5.1 using Havey Balls. The decision to utilize the Eclipse Milo Open Source framework for the PoC implementation has been made, since the framework implements the full OPC UA Server and Client API specifications and is actively supported by industrial partners.

6 Concept

This chapter will describe the concept of the PoC to be integrated to R'n'P. Section 6.1 discusses and list requirements for the infrastructure and architecture of the platform, the technology decisions taken for the PoC implementation as well as mandatory and optional features. In Section 6.2 will highlight the implementation approach of the PoC highlighting the way the service has been integrated into the platform.

6.1 Requirements

In this section summarizes and give an overview over the requirements, gathered from the analysis of the R'n'P platform and the discussion with the research partners. Section 6.1.1 will draw out the infrastructural requirements set on this work and the kind of deployment processes targeted. In Section 6.1.2 the architecture will be discussed into which the PoC should be integrated following by Section 6.1.3 which describes the technologies to use for the implementation and integration. With Section 6.1.4 and Section 6.1.5 this chapter will be concluded by providing mandatory and optional Use Cases for this work.

6.1.1 Infrastructure

In Section 2.1 it has been described, that the R'n'P platform is built upon services following the SOA approach, where each of the services takes responsibility for one specific domain task. Further, every service is separated into a business logic component serving the API for its functionality, and a persistence layer component implemented either as a SQL or NoSQL database. Following this pattern, this work deduces that the requirement will be to implement the PoC as a service, which stores the machine tool configuration data as well as the G-Code for the production part in a database and serves its API over a service layer component. Further, both components will be realized using Docker containers. The business logic container will register itself to the service discovery component and be routed over the API gateway.

6.1.2 Architecture

As shown in Section 6.1.1 the PoC to be realized will be packaged into a Docker container and its functionality will be split up into a persistence layer and a business logic layer. As the machine tool integration represents a part of the resource service process, it doesn't need a UI integration, as this part is already implemented in the R'n'P architecture.

More precise – the PoC should implement an API, which can be requested from the resource service, to trigger the production of the required parts contained in a customer order.

6.1.3 Technology

A core requirement to the PoC is the support of the OPC UA protocol. Over OPC UA the PoC should communicate with the machine tools already registered in the resource service of the platform. Second requirement is the filtering and recognition of G-Code data as it will be transferred to the PLC of the machine tool. For the integration to R'n'P, the service should support the communication of JSON over HTTP as well as the interaction with the service discovery server and the API gateway.

6.1.4 Mandatory Features

As result of this work, a PoC should be implemented and integrated into the R'n'P platform, so machine tools can be controlled over the Cloud platform. Therefore, a possibility is required to transfer G-Code data to the platform and from the platform down to the machine tools already provisioned. Finally, the possibility to trigger the start or the stop of a part production manually or automatically by the main production process is as well wanted. Figure 6.1 lists the emerged Use Cases and shows their relation to each other following the UML syntax and semantic described in [Obj17].

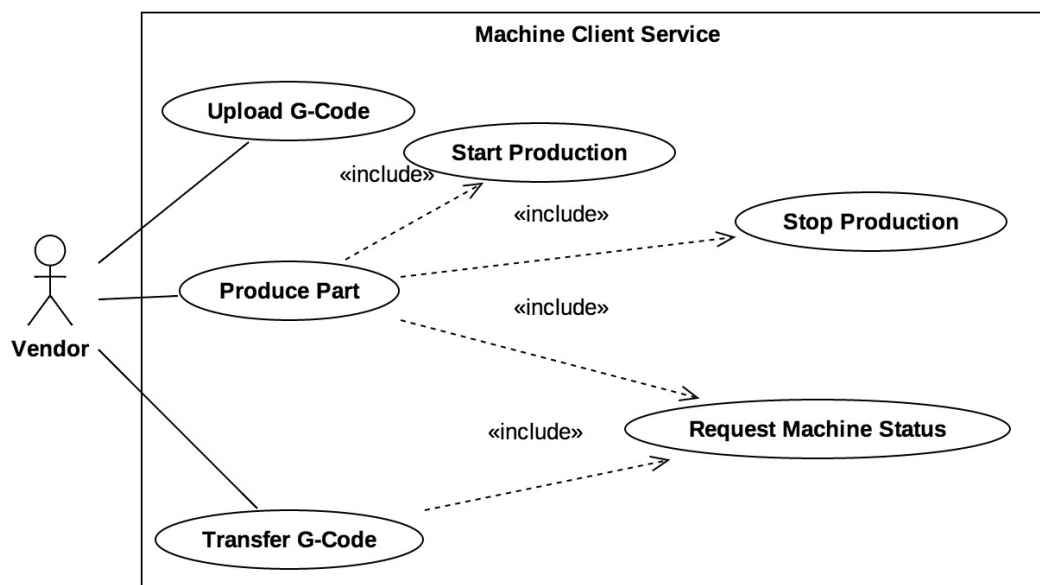


Figure 6.1: The Use Case diagram for the Proof-of-Concept

The following Tables 6.1 to 6.4) will describe the Use Case definitions for the ones shown in Figure 6.1. After the definition of the Preconditions for the given Use Case, an ordered description is following, by indicating the order of execution on the left side of the performing actors' description. The descriptions include only the regular process flow, as the side effects are described in more

detail in the specification, created during this research work. This work will highlight, that the definition of the Use Cases to start and stop the tool production have been united into the Use Case of the general part production as the steps are identically and only change in the control signal sent to the machine tool. Further, the exception and error handling of these both Use Cases is exactly the same, as in the general part production.

To start the manufacturing process of the production of a specific part, G-Code generated through a CAD-CAM chain by the user has to be uploaded to the Machine Client Service. Table 6.1 shows the Use Case description for this vendor task. The Upload is initiated through an HTTP-POST API call to the service, which then persists the G-Code and the machine tool configurations to its database, returning a success message in form of an HTTP 200 OK response to the vendor or services in general, calling the service API. After G-Code has successfully been uploaded to the platform, Table 6.2 describes the check of the machine tool status for further manufacturing processes. The check is described more abstract, as every supported call OPC UA API could be transmitted down to the machine.

By positive response on the machine tool status, the PoC can now proceed with the transfer of G-Code down to a connected machine tool for following part production. The transfer described in Table 6.3 shows the specific roles and actors in the transfer. The G-Code is transferred to the machine tool, by opening a *UA_FileType* on the machine tools OPC UA Server, writing the persisted G-Code on the client-side to the opened file and closing the writing process by linking the new *UA_Object* respectively with its surrounding *AddressSpace*. Finally, when all prerequisites are met (the machine tool status was checked and the machine tool is available, G-Code has been persisted to the platform and transferred down to the machine tool), the PoC can start the (or stop an already running) part production as presented in Table 6.4. Here, the preconditions need to be met are summarized, as well as the way, how G-Code production has to be started on the machine tool.

Table 6.1: Use Case description for the upload of production data

Name	Upload G-Code
Identifier	/UC-10/
Description	The vendor converted his CAD files into G-Code, so they can be transferred to the machine tool.
Actors	<ol style="list-style-type: none"> 1. A R'n'P platform user with the vendor role. 2. The OPC UA Client Service.
Trigger	The vendor has prepared the G-Code and wants to transfer it to the machine tool.
Level	User-View
Invariants	<ol style="list-style-type: none"> 1. The user stays logged in and keeps his OAuth2-Token with the appropriate scope. 2. The uploaded files have not been changed in format, type or content.
Preconditions	<ol style="list-style-type: none"> 1. The user is already logged in to the system using his web-browser and his session is assigned with an OAuth2 Token including the scopes <i>user</i>, <i>customer</i> and <i>vendor</i>. 2. The user has the G-Code to be uploaded. 3. The G-Code has the MIME-Type <i>application/x-netcdf</i>. 4. The user knows the identifier for the order, the G-Code is for.
1 User	The user uploads the G-Code using an HTTP-POST request to the OPC UA Client Service.
2 System	The systems persists the G-Code categorized under its order identifier and responds to the user with an HTTP-201 CREATED response containing order identifier and the production part identifier.
Postconditions	The G-Code has been persisted to the system. From the technical point of view, the service responds with an HTTP-201 CREATED Status. The user can download the file over the same API by providing the production part id. The G-Code file has not been manipulated after the upload.

Table 6.2: Use Case description for the retrieval of the current machine tool status

Name	Request machine tool status
Identifier	/UC-20/
Description	The service is able to request the current status of the machine tool using the OPC UA protocol. The service should only request, if a machine tool is prepared for production and does not deliver any error states.
Actors	<ol style="list-style-type: none"> 1. The OPC UA Client Service. 2. The machine tool to be requested.
Trigger	Use Cases including this Use Case have been triggered as shown in Figure 6.1.
Level	Technical View
Invariants	This Use Case describes only the retrieval of OPC UA specific information. As long as the protocol and correction are correct, this Use Case doesn't have any invariants.
Preconditions	<ol style="list-style-type: none"> 1. The OPC UA Client Service established and holds a connection to the OPC UA server of the machine tool. 2. The machine tool is available over Ethernet and OPC UA.
1 OPC UA Client Service	<p>The OPC UA Client Service requests one of the states from the machine tool:</p> <ol style="list-style-type: none"> 1. Is the machine tool currently in an error state? 2. Is the machine tool ready to produce a part (e.g., material prepared and machine tool loaded)? 3. Is the machine tool currently producing a part?
2 Machine-Tool	The OPC UA Server of the machine tool responds to the requests of the client service.
Postconditions	The OPC UA Client Service gathered the requested data from the machine tool. This data can now be used for further production process steps.

Table 6.3: Use Case description for transfer of production data to the machine tool

Name	Transfer G-Code to a Machine-Tool
Identifier	/UC-21/
Description	The vendor can transfer previously uploaded G-Code (as described in Table 6.1) to the machine tool.
Actors	A R'n'P platform user with the vendor role, the OPC UA Client Service and the machine tool the data should be transferred to.
Trigger	The vendor has uploaded G-Code to the system and wants to transfer it to a machine tool.
Level	User-View
Invariants	The transferred G-Code stays persisted in the database and will not be modified. The transferred data doesn't override data already existing on the machine tool control units.
Preconditions	The user is already logged in to the system using his web-browser and his session is assigned with an OAuth2 Token including the scopes <i>user</i> , <i>customer</i> and <i>vendor</i> . G-Code has been transferred as described in Table 6.1.
1 User	The user requests the Client-Service with an HTTP-POST method containing the identifiers of the part to be produced and the producing machine tool.
2 System	The Client-Service checks the requested machine tool for its current state as described in Table 6.2.
3 System	The Client-Service loads the G-Code for the part from the database.
4 System	The Client-Service creates an OPC UA connection with the Server on the machine tool and sends the G-Code data.
5 The Machine-Tool	The OPC UA Server on the machine tool receives the request, and stores the G-Code in the AddressSpace to a new file and loads it to the NC of the machine tool. Finally, a response, that the persistence was successful is sent back to the Client-Service.
6 System	The Client-Service informs the user, that the transfer was successful by responding with an HTTP-200 OK status.
Postconditions	The G-Code has been transferred to the machine tool and into its NC. The user is informed that the transfer was successful and the user can continue the usage of the platform. G-Code is now a new File in the OPC UA name space and can be used to produce the part.

Table 6.4: Use Case description for the production control of the machine tool

Name	Start and Stop a part production
Identifier	/UC-22/
Description	G-Code data has been successfully transferred to a machine tool. This G-Code should now be run to produce a part.
Actors	The OPC UA Client-Service of the R'n'P platform, as well as the control composite of the OPC UA Server, the PLC and NC of the machine tool as the second actor.
Trigger	G-Code has been transferred to the machine tool and now the part production should either be started or stopped.
Level	Technical View
Invariants	The G-Code transferred to the machine tool stays in the same location of the PLC and NC after the production is started or stopped. The machine tool stays connected to the R'n'P platform during the whole process.
Preconditions	The G-Code was properly transferred to the machine tool as described in Table 6.1. The OPC UA Server and the machine itself are connected with the R'n'P platform. The machine tool is not in an error state as described in Table 6.2.
1 OPC UA Client-Service	The Client-Service triggers the start or the stop of the production.
2 The machine tool	The machine tool loads the G-Code and produces the part or stops its current production, not going into an error state.
3 OPC UA Server	The server registers the processes of the machine tool and sends it to the message broker of the R'n'P platform over a response to the OPC UA Client-Service of the platform.
Postconditions	The part has been produced or the production process of the machine tool has been stopped. The current state of the machine tool can be monitored over the OPC UA Client-Service of the R'n'P platform.

6.1.5 Optional Use Cases

Besides the requirements defined to validate the PoC created in this work, several optional Use Cases are defined as well to extend the capacity of the PoC which could be implemented during this work, or providing inspiration for future research on this matter. The optional Use Cases pointed out during the research phase are described below in hastily manner giving space for requirement changes and other input and ideas coming from future research:

1. More than the control over the OPC UA Client-Service of the R'n'P platform developed during this work, one could image to transfer signals in form of visual information to the Human Machine Interface (HMI) of the machine tool. With this Use Case, workers at the production site of the machine tool are give the possibility to interact with the system and visually monitor the current state of the whole manufacturing process.
2. At the state of this research, the machine tool will be integrated only by pre configured data from the manufacturing. Allowing an auto-provisioning of the machine tools could increase the productivity of manufacturers and create a very homogeneous production environment.
3. One could imagine, that it could be useful to attach a monitoring or machine learning component to the service to be implemented. With a Big Data approach the created PoC could deliver more insights on the machine to the manufacturer increasing productivity and the whole manufacturing site performance.

6.2 Approach

With the requirements pointed out in Section 6.1 this work will now present the approach used to implement and integrate the desired PoC into the R'n'P platform. Using the defined requirements and Use Cases in Section 6.1 as well as relying on the research foundations highlighted in Chapter 2, the PoC will be tested against these. The developed PoC should be a service, independent of the R'n'P platform to ensure a possible use on every CM platform working with the OPC UA protocol.

6.2.1 Integration

As distilled from the platform analysis and defined in the requirements, the PoC will be packaged as a service, following the SOA principles into two Docker virtualization containers. One container will contain the business logic of the PoC and serve its REST-ful API. A second container will serve the SQL database using PostgreSQL¹ as the Relational Database Management System (RDBMS). The PoC itself will communicate with the R'n'P platform by registering itself to the service discovery of the platform and being routable through the API gateway already implemented as the edge entry point of the platform. The binding with the machine tool for the PoC will be done using an external URL of the OPC UA Server of the machine tool.

¹<https://www.postgresql.org/>

6.2.2 Target Architecture

As shown in Figure 6.2, the UML Component Diagram shows the PoC, which offers an API to the R'n'P platform to gather data and control the bound machine tool following the approach described in Section 6.2.1. The component diagram following the UML syntax and semantic described in [Obj17], shows that by adapting the SOA architecture of the R'n'P platform, the service can be integrated seamlessly. The OPC UA Service encapsulates the OPC UA functionalities into a single component, offering its API to interested R'n'P services. The machine tool is bound to the Cloud platform only by its OPC UA Server.

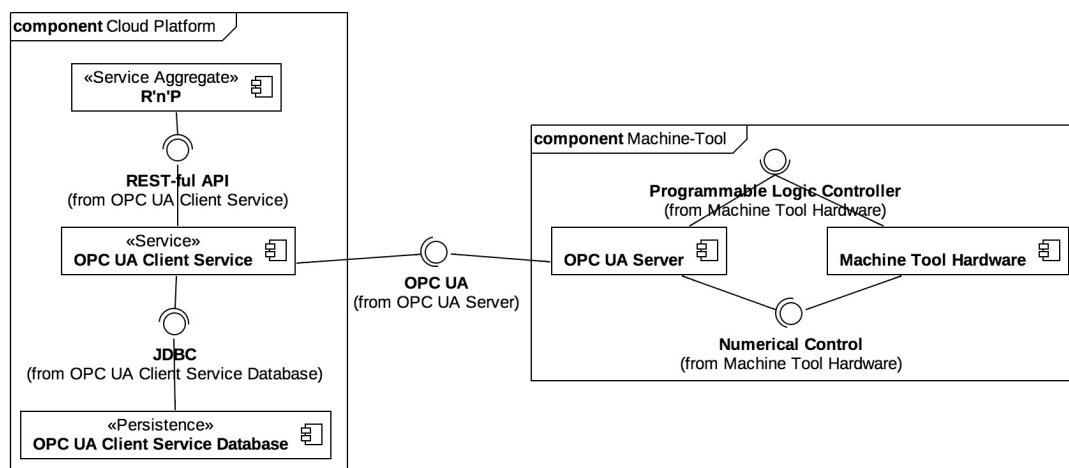


Figure 6.2: The component diagram of the integration of the Proof-of-Concept

As presented in Figure 6.2 PoC service communicates with and controls the machine tool only addressing its OPC UA server. The machine tool itself can be abstracted into three sub components: the OPC UA server running on the controlling industrial computer attached to the physical machine tool, the PLC and the NC as physical parts of the machine tool.

The OPC UA Server receives client requests and commands and transfers them to the PLC unit of the machine tool. This abstraction layer ensures, that standard requests and commands sent to the server, can be translated without the knowledge of the Client of specific machine tool configurations. By this approach this work refers to the research target of this work, to implement a machine tool independent solution as gathered from the research described in Section 4.3.

The PLC of the machine tool is responsible for the physical and electrical control of the manufacturing and machine tool processes. It can execute control commands like data gathering from the machine tool state, start- and stop of the production and transfer data to the NC. Processing of production parts data, planing of the milling routes for example and executing the part production is handled by the NC. Production data is loaded from and to the NC via the PLC and the current state is transferred to the PLC in real-time ensuring a correct monitoring of every production process.

Making use of the OPC UA specification, the PoC and the machine tool can be integrated without adding too much of abstraction layers keeping the architecture clean, understandable and maintainable without loads of effort.

6.2.3 Services

After describing the high level integration process, the services and APIs provided by the PoC will be highlighted in this section. Figure 6.3 shows out of which services and what kind of services, the PoC will be implemented. The content presented in Figure 6.3 is described using a UML Component Diagram following [Obj17].

Each part of the functionality will be bound into a four tier composite – the Entity representation of the domain specific objects to be persisted to the database, a Repository layer making database queries available using the Spring Boot Java Persistence Application Programming Interface (JPA) of the Entity, a Service layer implementing the business logic of the specific functionality and finally a Spring Boot REST Controller wrapping the Service layer functionality into a REST-ful API also handling the mapping of JSON objects to Java [GJS+15] Entities.

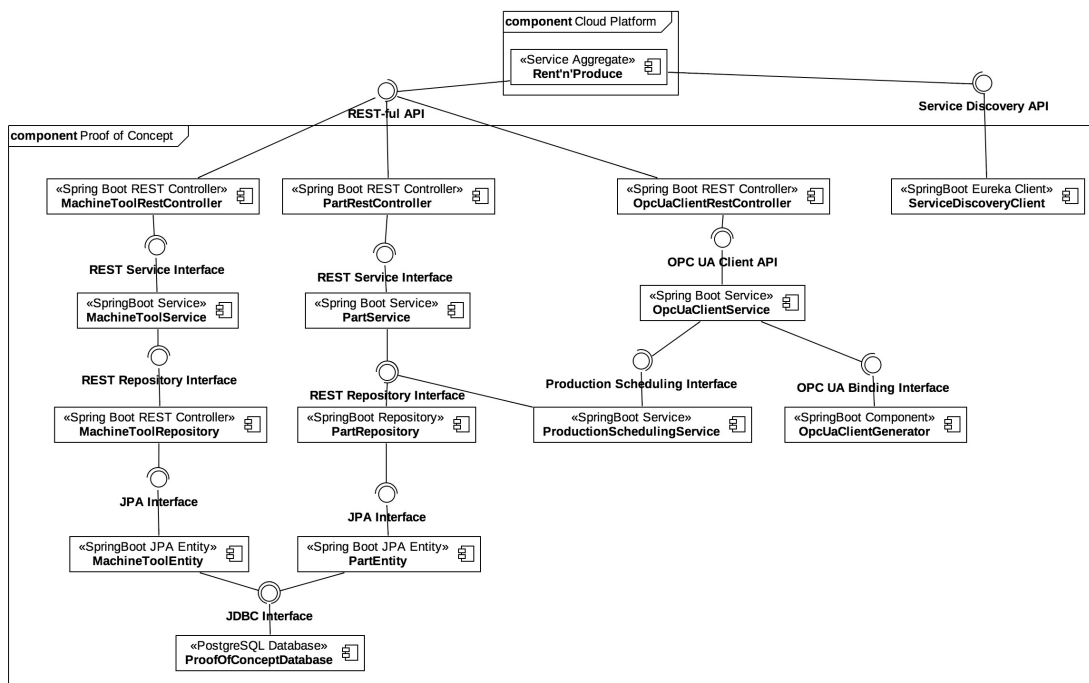


Figure 6.3: The component diagram describing service details of the Proof-of-Concept

On the left side of Figure 6.3 the described blocks serving the APIs to persist, retrieve and manipulate data of the production part and machine tool domain entities are presented. These APIs are responsible for persisting new machine-tools and configurations to the system as well as the persistence of the parts to be produced with their G-Code data and production dates to allow automatically or manual scheduling of the production processes. As the Service Discovery and the API gateway of the R'n'P produce are tightly bound, offering the API to the Service Discovery, the PoC is automatically able to be routed through the API gateway, discovered and requested by other services of the platform using both, the API gateway and the Service Discovery, and to be auto-scalled by the underlying Docker Engine and the deployment with the Docker-Compose Container orchestration mechanisms.

Further, Figure 6.3 presents on the right side the core functionality of the PoC. The Service Discovery Client binds the PoC to the Service Discovery Server of the platform, allowing to be seamlessly integrated into the platform. The OPC UA Client Service implements the connection to the machine tools, retrieved from the part data and the machine tool Entity representation. On the other hand, the Client Service creates the connection, using the provided Client Generator, implementing a generic connection utility for security credentials, machine-tool data and further required information to establish a clean connection. Finally, the OPC UA Client Service can produce parts automatically, by triggering the Scheduling Service to retrieve the production date and time of the part to be produced next, and comparing to the system time, triggering the automatic start of a part production as the machine tool is available and its state is not an error state. The Service functionality is made public by a single API wrapper. It should be noticed, that the Client Service functionality is implemented in an asynchronous manner, as data transfer and part production can take quite a long time and the client should not wait so long to inform the user about a success or a failure. Therefore, the client will respond to the user at the moment, as all paradigms for a successful file transfer or production start are given. After the HTTP response, the user is informed over the event messages, sent to the message broker, available at global scope for the entire R'n'P platform. This approach ensures a user experience with no deadlocks and at the same time paralleling production processes.

6.2.4 Workflow Concept

Targeting the goal to be able to upload G-Code to the R'n'P platform, to transfer the G-Code to a connected machine tool and to produce the production part described with the G-Code, the general workflow of the whole part production of the PoC using the OPC UA protocol will now be described. The workflow is described using a UML Sequence Diagram in Figure 6.4 as specified in [Obj17].

Figure 6.4 shows the interactions between the following four actors:

1. The *Vendor*, representing a manufacturer wanting to start or stop a production workflow.
2. The *OPC UA Client Service* (the PoC) integrated into the R'n'P platform.
3. The *OPC UA Server* attached to the machine tool, which should produce the required part.
4. The *OSACA*, a real physical machine tool provided by the ISW for the scope of this thesis.

User interactions with the PoC are realized by triggering the REST-ful API provided by the PoC using JSON over HTTP as communication base. System information related to other components of the system and to the user are implemented using asynchronous messaging with the Advanced Message Queuing Protocol (AMQP) provided by the global message broker of the R'n'P platform. Communication between the system and the machine tool, which should produce the part, is managed by the OPC UA protocol as described in Section 2.4.

This work refers to the basic workflow of the PoC as the implementation of the Use Cases to upload G-Code, request the current machine tool state, the transfer of G-Code to the machine tool and the start of the part production, as shown in Tables 6.1 to 6.4.

Figure 6.4 describes the first steps of the process of the Use Case for G-Code transfer shown in Table 6.1, in which the user has G-Code prepared from the CAD file of the part to be produced.

The user sends an HTTP-POST request in JSON format to the system, containing his authentication header, the G-Code to be uploaded to the system, the identifier of the machine tool that should produce the part and the production time of the part. It should be mentioned, that by sending the identifier of a selected machine tool to produce specific part, the service checks if the catalog contains multiple machine tools with the same configuration parameters. If this is the case, one machine tool that is available at the production time of the part will be selected to produce the part.

Receiving the user request, the service checks if the request is valid and if the user has the required permissions to perform the API call. If the requirements are not met in the request it will be rejected and the user gets an information in the API response. Assuming that the request has been sent correctly, the service starts to check the machine tool catalog and the configurations of the production process.

If the configuration check and the following persistence of the production part and the machine tool connection (referring to an OPC UA connection to the OPC UA Server of the machine tool) is successful, the user request is finished with a positive response and the automatic production process starts.

By starting the automatic part production, the PoC checks every second, if the order of the parts to be produced is still consistent with production dates of all valid parts in the database. Is the current system time 30 minutes away from the start of the production, the system begins to transfer the G-Code data of the part to the machine tool by firstly checking the current machine status (not including that the machine tool can be in production mode) as shown in Table 6.2 and begins to transfer the G-Code data to the machine tool as described in Table 6.3.

In the case, that the production paradigms are not met, the part will be rescheduled in the database to a deferred state and the manufacturer user is informed over an asynchronous message. In the other case that, all requirements are met, the production is started at the moment the system times equals the part production time. After the successful production of the part, the user is again informed over an asynchronous message.

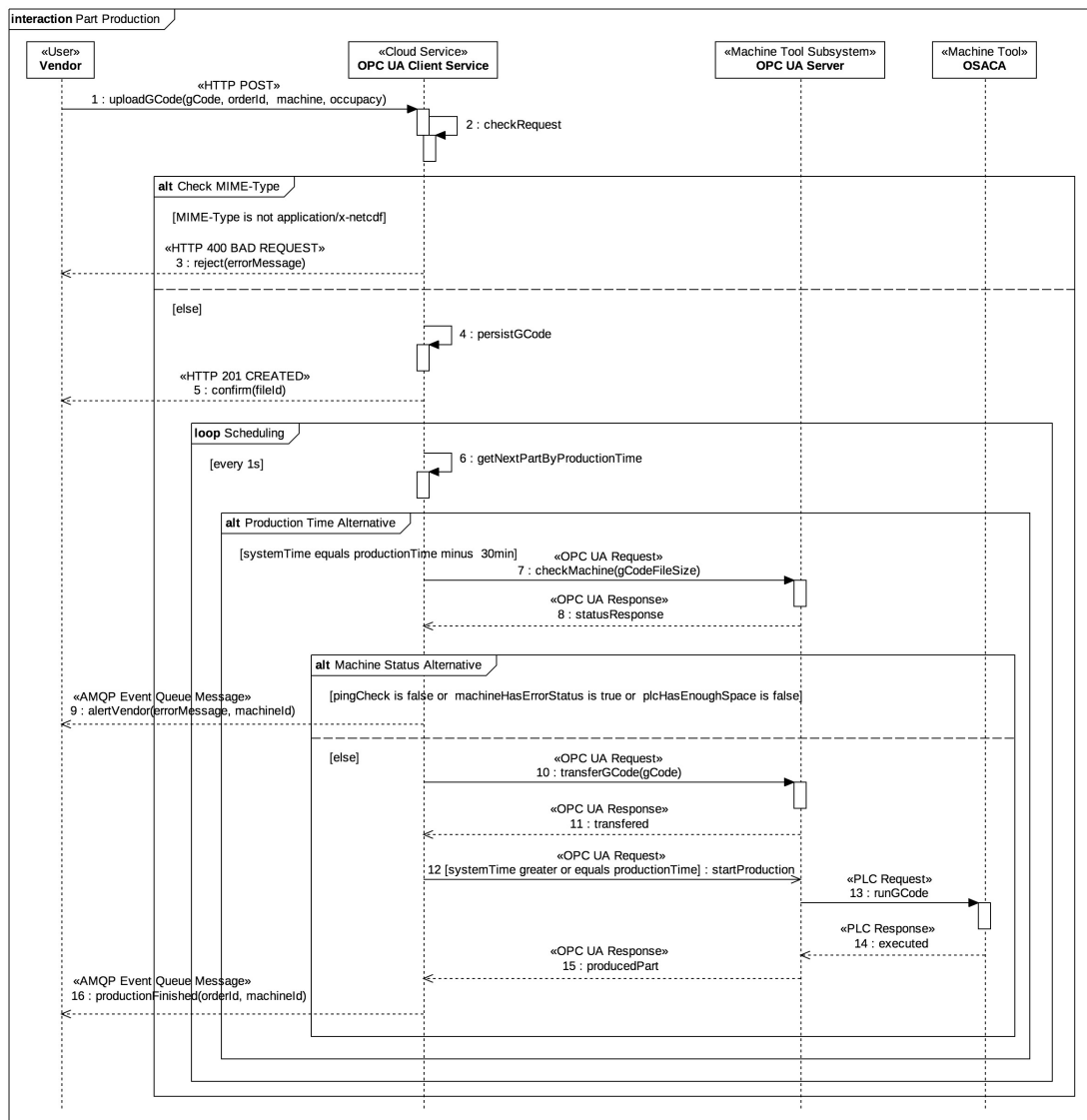


Figure 6.4: Sequence diagram of the general part production process

By describing the workflow shown in Figure 6.4 all the requirements and Use Cases described in Section 6.1 have been handled. Further, this work highlighted that with the prior knowledge of the concepts described in Chapter 2 and the analysis of the system, it is possible to model an automatic manufacturing process only relying on Open Source Technologies and the OPC UA specification. In Section 6.2.5, the machine tool integration between the R'nP OPC UA Client Service running in the Cloud and the machine tool itself will be described to finalize the whole manufacturing process model.

Nevertheless, it should be mentioned that in the scope of this PoC, the production workflow rely on the provided OPC UA functionalities of the Beckhoff TwinCAT V3 OPC UA server provided by the ISW for this work. It is further assumed, that the provided functionalities can be triggered as described in [Bec18] by only executing the appropriate OPC UA Client API calls from the client

to the server side. As result, this work has to notice, that only under the case, this assumption is correct, the interactions between the *OPC UA Client Service*, *OPC UA Server Service* and the machine tool *OSACA* actors, can be successfully implemented. Only under these assumptions a PoC can be implemented, without taking any changes on the machine tool side and ensure a machine tool and control unit independent manufacturing approach in the PoC.

6.2.5 Machine Tool Integration

In Section 6.2.4 this work presented the concept of the regular workflow for the automatic part production of the PoC. The description of the machine tool integration side will finalize the model. The machine tool will be integrated using its OPC UA server.

To check the machine tool status, the client will read the current status data from the servers *UA_AddressSpace*. These variables are set machine tool specific and have to be determined for each server and machine tool. For the PoC, the Beckhoff TwinCAT V3² control will be used and OPC UA server for demonstration and testing purposes and describe machine tool specific commands using its specification defined in [Bec18]. The machine tool for experiment purposes is a 3-axis milling machine provided by the ISW, called *OSACA*.

The machine tool status check is handled, using the standard OPC UA client read function to determine, if the current machine state in the TwinCAT NameSpace is not of type *AlarmType* in the *UA_StatusCode*. If no entry is currently provided for any of these types (described in Clause 8.2 of [Bec18]) under the *DeviceState* OPC UA Object, the machine tool has no error state and it leaves us to check, if the machine tool is ready for part production or currently producing a part. The check if the machine tool is currently producing a part, is done by checking, if any motion state changes arrived in the *UA_NodeGetHandleList* as if the production is ready, the event list is empty. Connectivity analysis of the machine tool is handled implicitly as every read request of data requires a valid OPC UA connection.

For read and write operation calls sent by an OPC UA Client to an OPC UA Server, the OPC UA protocol specifies a standard *UA_FileType* [OPC17c] which should be implemented by the OPC UA server for the specific control used by the machine tool. The *UA_FileType* is represented by an array of bytes sent over the OPC UA connection. For the purpose of this PoC, the data sent during the connection is only encrypted using the OPC UA security mechanisms described in [OPC17a] to avoid encoding and decoding complications at the testing of the PoC.

Transferring of a G-Code data file is therefore handled by the PoC as follows:

1. The Client connects to the Server and checks if a program with the file name of the G-Code data already exists.
2. If the G-Code already exists, the client appends temporary the system date to the file name and triggers the creation of a new *UA_FileType* in the programs folder for the project.
3. After the empty file is created on the Server side, the client writes a byte stream into the file created using an *UA_Write* command to the server.

²<http://www.beckhoff.de/twincat3/>

4. Finishing the writing command, the client triggers the naming of the file and the setting of project specific parameters.

In the last step of the part production after successfully transferring the G-Code down to the PLC of the machine tool using the OPC UA server, the PoC can now start the part production at the required time. Therefore, the OPC UA Client Service of the glspoc triggers the OPC UA server to create a new *Task* in the TwinCAT automation project and execute it using the predefined workflow in *UA_Method* and *UA_Function* calls in the PLC.

It should be mentioned that for a machine tool independent implementation, the OPC UA server configurations and provided methods have to be homogeneously distributed over all machine tools connected with the platform. More precise information and a deeper discussion on that point follows in Section 7.5.

6.2.6 Security

Besides the implementation of an automatic part production from the Cloud, it should be mentioned that business relevant data passes the processes from to the Cloud, and down to the machine tools of manufacturers, which leads us to the need of securing the data transfer and providing some kind of user access and permission control. As the field of security is far beyond the scope of this work, some security approaches taken to secure the platform and the developed PoC only by touching the huge field of security in the manufacturing environment should be presented in the following.

First of all, the Client service in the Cloud should check, if the data uploaded is really G-Code on no other format. Therefore, a G-Code parsing library is used to check the content of the G-Code data and to prove, that no other parameters are set. Further, a header check is required to determine the file format for parsing. If not each of these criteria is given, the file upload is aborted, ensuring no harmful data can be uploaded to the Cloud.

User access and permissions to access the API is regulated by the Oauth2 protocol to handle user access sessions to the system. Only users with the appropriate access tokens can send requests to each service of the R'n'P platform, including the PoC. Besides that, only the manufacturers identified by their tokens, can see the machine tools they own and trigger the data transfer to them, which builds an additional security layer.

Finally, the access on machine tool level is secured using the OPC UA security mechanisms. Each OPC UA server of each machine tool has a fine granulated access and method call description secured using identity certificates. The PoC initializes the certificate negotiation at the first time, the machine tool is bound to the cloud platform. The service itself can only add G-Code files to the machine tool PLC but can't remove data from or interrupt native PLC implementations.

6.3 Conclusion

This chapter introduced the concept for integrating a PoC into the R'n'P CM platform. First, the R'n'P platform was analyzed and infrastructural, architectural and functional requirements for the PoC were distilled from the analysis made beforehand. The architecture for the PoC has been described as a SOA. The PoC will be packaged into a Docker container, bound to a second Docker container containing the SQL database serving as persistence layer for the PoC and will provide its functionality with a REST-ful API. The PoC will finally be bound to the OPC UA server of the machine-tool to be integrated only relying on the OPC UA protocol. This chapter described the binding process in more detail. Subsequently, Use Cases have been divided into mandatory and optional ones, which then have been described in more detail to ensure later comparison of the implemented PoC against them. Finally, basic security concepts and suggestions concluded this chapter by suggesting the usage of the OAuth2 authentication protocol for securing the PoC API and the utilization of the security mechanisms already implemented in the OPC UA protocol.

7 Discussion

In this chapter, this work will compare the concept and approach described in Chapter 6 to the actual implementation and experimental work with the PoC. Section 7.1 summarizes the motivation and goals of this work. In Section 7.2, a recapitulation over the procedure for implementing the goals set is taken. Section 7.3 will provide the results of how well the PoC could be integrated into the R'n'P platform. Comparison of the self defined Use Cases and goals set for the scope of this work with the actual implementation will be described in Section 7.4. Finally, limitations of the PoC and the whole approach are shown in Section 7.5 closing this chapter.

7.1 Main Objective

The R'n'P platform presented in Section 2.1 aims to model and simulate the whole CM process, described in Section 2.3; from ordering manufacturing resources, the planning and scheduling of production processes, to provisioning and controlling physical machine tools from the Cloud platform. As R'n'P lacked the functionality of communicating and transferring production data down to bound machine tools as shown in Section 2.1.1, this thesis aimed to model, specify and integrate a PoC to provide the platform with exactly these functionalities. In the context of research related to this work as highlighted in Chapter 4, this thesis focused on implementing a PoC that is machine tool independent and only relying on the OPC UA communication protocol trying to show, that by using OPC UA, machine tool and CM platform independent control over the Cloud is possible and applicable to manufacturing sites. Described in more detail in Section 2.4, this work refers to OPC UA as the machine tool abstraction architecture, abstracting machine tool control, security and operational signals, in a unified architecture by realizing an architecture, where the Server resides on the machine tool and executes commands from the clients. This principle is often described as an inverted client-server communication system.

The functionality of the PoC had to be tested with a productive manufacturing unit provided by the ISW for this work. The motivation of this work was to transfer production part descriptions in G-Code format down to the machine tool and start the part production, only relying on the OPC UA protocol.

Since the R'n'P platform has been developed by using the concepts and technologies described in Section 6.1.2, the approach for the integration has been the same way, as presented in Section 6.2. This resulted in the usage of technologies and concepts described in Chapter 3. As a result, a Spring Boot Service was implemented using the Java programming language, virtualized and encapsulated into a Docker container and integrated into the SOA architecture providing its functionalities by a REST-ful API. Further, OPC UA related functionalities and abstractions have been realized using the Eclipse Milo Java Framework. On the basis of the developed requirements and set goals, the PoC had to be substantiated with a concept, which is shown in Chapter 6.

7.2 Approach

To achieve the objective recapitulated in Section 7.1, the steps described in Chapter 6 have been evaluated and executed. The evaluation and integration processes of the PoC have been divided into Section 6.1 and Section 6.2.

Section 6.1 distilled the relevant requirements and features to be implemented into the PoC by analyzing the initial state and functionality of the R'n'P platform. Use Cases have been defined to clear the scope of the PoC as well, as to ease the development process of the PoC.

The approach of the implementation of the PoC has been described in Section 6.2. Here, integration details and the target architecture have been described. A detailed presentation of the components included in the PoC as well as the machine tool communication workflow using OPC UA has been given in Sections 6.2.3 to 6.2.5 as well as the binding of the machine tool to the R'n'P platform. The resulting PoC has further been tested against a Beckhoff TwinCAT V3 OPC UA Server and PLC. Finally, basic security implementations included into the PoC have been described in Section 6.2.6.

7.3 Integration Results

After implementing the PoC with the requirements defined in Section 6.1 and the goals of this work listed in Section 1.2 in mind, the developed PoC could be totally integrated into the R'n'P platform as planned. By using the Docker container runtime, the PoC could potentially be integrated into any PaaS solution and architecture, running on the Docker Engine as container and orchestration platform.

The implemented REST-ful API encapsulates all functionalities of the PoC and provides it to any third-party service, that is able to communicate using JSON over HTTP and receive messages from an AMQP message broker. The database serving the PoC service can be interchanged to any SQL database as well as additional components like Caching, Searching and Logging utilities can be added by accessing the Spring Framework API and its provided integration functionalities.

As the OAuth2 protocol is provider independent, custom implementations of a user access and permission management system can be added or replace the current implementation. Same criteria were met for the service discovery and API gateway Cloud Computing patterns as described in [Ley11]. The OPC UA functionalities of the service are encapsulated to the service itself, providing an extensible and generic OPC UA Client Service supporting the OPC UA protocol only limited by the functionalities provided by the Eclipse Milo Client Library implementation.

Summing up the results, the PoC could be implemented and integrated seamlessly to the R'n'P platform. With this result, the PoC fulfills the first four abstractly defined goals, shown in Section 1.2, that the PoC should be platform independent and serving a standard API.

7.4 Use Case Implementation

After defining the Use Cases in Section 6.1.4, this work will prove each of them against the actual implementation of the PoC. This will serve as a proof for the concept and unveil limitations of the concept presented in Chapter 6.

7.4.1 Uploading of Production Data

By offering the API to upload simple files in the G-Code file format, this Use Case is proofed by the PoC. Files can be uploaded and before persisting the files for a specific production step, the system checks, if the header parameters for the file type and the file content itself is G-Code compliant. Requests with any other type of data will be rejected. Exception handling was implemented to handle unauthorized requests, wrong request contents and unspecified parameters as well as requests to API entry points not containing the requested data. It should be concluded, that the PoC implements the Use Case as described in Table 6.1.

7.4.2 Machine-Tool Status Checking

The implemented PoC provides an API for checking the current machine tool status. Behind this API, a generic OPC UA connection utility generates OPC UA connections for the provided machine tools already persisted in the database beforehand. The API implements an enumeration-based approach for different connection security requirements, to provide granulated access to the specific values requested by the user. Anonymous connections can be established, as well as connection using specific *Username* and *Password* credentials or Client-Server Certificate pairs.

An asynchronous connection is built each time the OPC UA API of the PoC is triggered, checking the following parameters:

1. Is the machine tool available over Ethernet?
2. Is the machine tool available for a connection over OPC UA?
3. Is the machine tool currently in an OPC UA error state?

As the last parameter is machine tool specific and relies on the OPC UA server implementation on the machine tool; in the scope of this PoC only the functionalities provided by the TwinCAT V3 control unit used in this work and specified in [Bec18] are referenced. More details on this machine tool specific approach will be discussed in Section 7.5. Nevertheless, the PoC provides the API as described in the Use Case shown in Table 6.2, which leads us to the conclusion, that the PoC provides the functionality required. This statement is argued by the evidence, that the PoC implements this functionality in total accordance to the OPC UA Client API and the OPC UA Server API of the TwinCAT V3 specified in [Bec18], as described in Section 6.2.5.

7.4.3 Production Data Transfer

With the possibilities implemented in the PoC, as described in Section 7.4.2, the machine tool status can be checked, before a previously uploaded G-Code file can be transferred to the machine tool to produce the part. The approach described in Section 6.2.5 is the standard process to complete a file transfer following the OPC UA specification respectively to the TwinVAT V3 specification [Bec18] used as control unit for the PoC.

After monitoring and logging the requests sent in each step of the transfer workflow described in Figure 6.4, this work ensures that the implemented functionality is correct on the side of the OPC UA Client Service API, as it reflects the suggested OPC UA process and data flow. An interesting recognition taken from the experiment to transfer G-Code data to the TwinCAT V3 PLC has been, that even if the TwinCAT V3 offers the described API to handle OPC UA *UA_FileType* requests, the OPC UA server of the machine tool responded, that the requested function is not available and returns a *UA_StatusCode* showing a correct request but bad response.

This issue has been further analyzed down to the recognition, that the OPC UA Server side cancels, the request due to *UA_UserWriteMask* permission problems. Further, this work revealed from the TwinCAT V3 specification [Bec18], that the read and write permissions, can only be manipulated on the Server side of the connection, making it impossible for third-party applications like the PoC to manipulate access and permission hierarchies.

Since the permission and write-access issues unveiled the fact, that the implemented PoC is not capable to manipulate *UA_AddressSpace* permissions and hierarchies on the TwinCAT V3, further tests have been processed to find out, whether the PoC implementation would work on a different OPC UA Server implementation. Therefore, a Prosys OPC UA Simulation Server¹ was bound to the PoC, not changing the configurations; neither of the Simulation Server, nor the ones of the PoC to test the implemented functionalities and unveil potential implementation errors. Surprisingly, it was noticed that the PoC could create a new *UA_FolderNode* in the *UA_AddressSpace* of the OPC UA Simulation Server, further creating and opening a new file, writing the G-Code content into it and finally persist it on the OPC UA Server side. After further investigation it was found, that the Prosys Simulation Server allows the creation of files and manipulation of the *UA_AddressSpace Objects*, without further specifying user permissions. The tests worked as expected for each kind of connection: anonymous connections, certificate-based authentication connections and user-specific connections.

These results led us to the conclusion, that the functions implemented in the PoC, as well as the OPC UA function calls themselves have been implemented correctly. It is concluded that the permissions and function call implementations on the OPC UA Server side have to be configured beforehand and provisioned to the PoC, so afterwards manufacturing processes can be modeled. This realization contradicts the concept to create a PLC and machine tool independent service for file transfer and production. The hereby unveiled limitations will be discussed in more detail in Section 7.5. In summary, this work has shown with the PoC, that it is possible to transfer G-Code data for part production using the OPC UA protocol.

¹<https://downloads.prosysopc.com/opc-ua-simulation-server-downloads.php>

7.4.4 Part Production

As described in Section 7.4.3, the transfer of real production data in G-Code format, down to the machine tool provided for testing, could not be completely tested. Nonetheless, the PoC implements the functionality to read and set status data on the TwinCAT V3 PLC using its OPC UA Server, according to [Bec18]. As expected, these data were available to be read and written over OPC UA and the *UA_StatusCodes* and *UA_Types* could be read, handled complications.

However, it was as expected as well, that the PoC was not able to write and set the values, as needed to send a production start or production stop signal. Once again, the TwinCAT V3 expects previous server configurations to allow data manipulation on the server side.

Additionally, the TwinCAT V3 environment expects, that G-Code data for part production, is present on the system in the context of the integrated development and production environment on the TwinCAT V3, which means, that besides a potential transfer of G-Code data using OPC UA from the PoC to the OPC UA Server of the TwinCAT V3, the data has to be loaded into the NC part of the TwinCAT V3 control. Even with the fact, that the TwinCAT V3 OPC UA API provides predefined *UA_Functions* for these operations, the G-Code files have to be properly linked beforehand in the *UA_AddressSpace*, which underlies several permission restrictions and security limitations, making it impossible for the PoC to accomplish this manufacturing process task.

Assuming, that the parameters could be read and written under the fitting *UA_UserWriteMask*, the expectation was made that with a proper configuration of the OPC UA server of the PLC the start and stop of the production, would simply trigger already implemented functions on the PLC to pursue with the part production. However, with the issues described in Sections 7.4.3 and 7.4.4, the last of the five goals set in Section 1.2 could not be totally fulfilled.

7.5 Limitations

As a result of the test of the PoC against the Use Cases and requirements define beforehand, it was found out that the requirements could not be totally met. Although the PoC have been seamlessly integrated into the R'n'P platform, a fully machine tool and control unit independent functionality could not be implemented. Even with the fact, that the PoC could provide the functionality to integrate OPC UA Client API functionality into itself and establish OPC UA connections in a generic manner, restriction to the transfer of control signals down to the machine tool have been discovered.

In Section 7.4.3 the issue was described that the handling of *UA_FileType Objects* depends on the provided functionality of the OPC UA Server of the machine tool. Further, the provided functionalities can vary between different Servers, even if they implement the same OPC UA Server API specification. More precise, even if the Server side implements the handling of the necessary events to transfer a G-Code file, the permission hierarchy on the server-side has to be configured, before the machine tool is bound to the PoC. This leads us to the fact, that the PoC has to be able to recognize provided functionalities of the Server side by iterating the OPC UA Server *UA_AddressSpace* as well as to have the permission to execute *UA_AddressSpace* manipulations.

Further limitations have been found during the research phase of the project. Dependent on the real-time requirements of the system, it could be possible, that the PoC doesn't fit these requirements

as a data, signal and file transfer to the control units can take a while, missing the required real-time signal slots. Further, the PoC totally depends on a prior knowledge of the OPC UA addresses and security configurations from the manufacturer as that these have to be provided during the persistence of machine tools to the R'n'P platform. Finally, an automatic provisioning of the machine tools by their OPC UA addresses is still not possible, as the types, versions and software states of the manufacturers PLCs can be highly heterogeneous, making it impossible for the PoC to know which kind of services the PLC really supports, making the need of a kind of gateway inevitable.

7.6 Conclusion

In this chapter, the results of the testing of the implemented PoC against the previously defined requirements and the concept described in Chapter 6 were presented. For the integration of the PoC into the R'n'P platform the results showed, that the PoC specified in the concept could be integrated seamlessly into the platform. Further, the requirements of the Use Cases to upload, persist and schedule production data as G-Code as well as the request of the current machine tool status could totally be met by the implemented PoC.

As described in Sections 7.4.3 and 7.4.4 limitations and negative results have been revealed. It was noted that the transfer of production data to the connected machine tool depends heavily on the implementation of the OPC UA server attached to it. For this reason, an automated and machine tool independent production from the Cloud could not be simulated on provided physical machine tool. Nevertheless, it could be shown that the PoC on the OPC UA Client API side correctly implements the OPC UA protocol and that these tests after; a specific adaptation to the OPC UA Server of the machine tool would have turned out positive. Finally, a closer discussion in Section 7.5 on the exact background of the problem was led.

8 Conclusion and Future Work

This Chapter provides a short summary of this work in Section 8.1. In addition, future work on the system is proposed and possible Use Cases are discussed in Section 8.2.

8.1 Conclusion

In this work, an approach for the automated part production in the manufacturing field using the R'n'P platform and the OPC UA specification has been proposed. To accomplish this task, a concept has been derived from the analysis of the R'n'P CM platform and its missing functionalities at the beginning of this work.

First, foundations of CM and the OPC UA specification were presented. This provided a base for examining the state of the art, the state of the research and related work. With the aid of observations made in foundational as well as related work, an approach implementing a working PoC which seamlessly integrates into the R'n'P platform was presented. The PoC encapsulates its functionality in a Docker container and provides it through a REST-ful API to be platform and language independent. Further, persistence and management models were developed and integrated into the PoC. The OPC UA and machine tool integration was implemented following the principles of the OPC UA specification. It has been found out, that even if the Cloud-based client implements the specified OPC UA functionalities, it is still a matter of machine tool configuration, making a truly machine tool independent solution not possible with this PoC. Finally, this work summarized the limitations of the PoC and the approach as well as providing information about the architecture and implementation for future research work.

8.2 Future Work

The presented approach is not totally machine tool independent, as it relies on the server-side implementations provided as OPC UA interfaces. For a better handling and analyzation of manufacturing sites topologies, it is suggested to provide a supplement service for *scanning* of the manufacturing sites, which should allow the service to get insights to the individual topologies. In addition, the PoC currently expects from the users, that they exactly know the addresses and configuration details of their machine tools. One could image to make research on manufacturing site provisioning and automatic registration of machine tools to the platform with ideal information about the possible manufacturing processes. At last, future research could extend the current R'n'P platform and especially the PoC, to implement a central authorization and certificate management service for all machine tools of a production site. This would ease the configuration and access management of each machine tool and lead to a higher security level and usability of the whole platform.

Bibliography

- [AIM10] L. Atzori, A. Iera, G. Morabito. “The internet of things: A survey.” In: *Computer networks* 54.15 (2010), pp. 2787–2805 (cit. on p. 13).
- [AWHM17] G. Adamson, L. Wang, M. Holm, P. Moore. “Cloud manufacturing—a critical review of recent development and future trends.” In: *International Journal of Computer Integrated Manufacturing* 30.4-5 (2017), pp. 347–380 (cit. on p. 31).
- [BB08] R. Battle, E. Benson. “Bridging the semantic Web and Web 2.0 with representational state transfer (REST).” In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6.1 (2008), pp. 61–69 (cit. on pp. 28, 29).
- [Bec18] Beckhoff Automation GmbH & Co. KG. *TwinCAT TC3 OPC UA Documentation*. Ed. by Beckhoff Automation GmbH & Co. KG. Mar. 7, 2018. URL: http://download.beckhoff.com/download/document/automation/twincat3/TF6100_TC3_OPC-UA_DE.pdf (cit. on pp. 14, 53, 54, 59–61).
- [Ber14] D. Bernstein. “Containers and cloud: From lxc to docker to kubernetes.” In: *IEEE Cloud Computing* 1.3 (2014), pp. 81–84 (cit. on p. 27).
- [BFKR14] M. Brettel, N. Friederichsen, M. Keller, M. Rosenberg. “How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective.” In: *International Journal of Mechanical, Industrial Science and Engineering* 8.1 (2014), pp. 37–44 (cit. on p. 21).
- [CJOC10] G. Cândido, F. Jammes, J. B. de Oliveira, A. W. Colombo. “SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications.” In: *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. IEEE. 2010, pp. 598–603 (cit. on p. 22).
- [CMC+17] S. Chen, R. Ma, H.-H. Chen, H. Zhang, W. Meng, J. Liu. “Machine-to-machine communications in ultra-dense networks—a survey.” In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1478–1503 (cit. on p. 31).
- [Ell16] C. Ellwein. *Rent’n’Produce: Secure Cloud Service for the Commissioning and Control of Production Systems*. Ed. by I. for Control Engineering of Machine Tools, M. Units. Aug. 3, 2016. URL: <http://www.isw.uni-stuttgart.de/forschung/projekte/rent-n-produce/> (cit. on pp. 13, 18).
- [ER17] C. Ellwein, O. Riedel. “Rent’n’Produce: A Social Media Platform for Cloud Manufacturing.” In: *Smart Materials and Nanotechnology in Engineering*. 4th International Conference on Smart Materials and Nanotechnology in Engineering (SMNE 2017). (Sofia, Bulgaria). Ed. by G. Lee. Vol. Lecture Notes in Earth Sciences. Lecture Notes in Earth Sciences. Information Engineering Research Institute. Bellflower, USA: IERI & PRESS, 2017, pp. 75–81. ISBN: 978-1-61275-526-7. DOI: 10.5729/lnes.2017.7.75 (cit. on pp. 17, 19, 20, 32).

- [Erl08] T. Erl. *Soa: principles of service design*. Vol. 1. Prentice Hall Upper Saddle River, 2008 (cit. on p. 18).
- [FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer, 2014 (cit. on pp. 19, 20).
- [FT00] R. T. Fielding, R. N. Taylor. *Architectural styles and the design of network-based software architectures*. 2000 (cit. on p. 28).
- [GJS+15] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley. *The Java Language Specification Edition 8*. Tech. rep. Oracle America, Inc., Feb. 13, 2015 (cit. on pp. 27, 50).
- [Had06] T. Hadlich. “Providing device integration with OPC UA.” In: *Industrial Informatics, 2006 IEEE International Conference on*. IEEE. 2006, pp. 263–268 (cit. on p. 22).
- [HBM+18] H. S. Haridas, S. W. Booker, P. F. McLaughlin, A. Watson, J. A. Strilich, J. Schreder. *Cloud-based control platform with connectivity to remote embedded devices in distributed control system*. US Patent App. 15/251,815. Mar. 2018 (cit. on p. 32).
- [Hei05] W. Heinrichs. “Do it anywhere.” In: *Electronics Systems and Software 3.4* (2005), pp. 30–33 (cit. on p. 13).
- [HL95] W. L. Hürsch, C. V. Lopes. “Separation of concerns.” In: (1995) (cit. on p. 35).
- [HS14] R. Henßen, M. Schleipen. “Online-Kommunikation mittels OPC-UA vs. Engineering-Daten (offline) in AutomationML.” In: *Tagungsband Automation 2014* (2014), pp. 59–74 (cit. on pp. 22, 23).
- [HX15] W. He, L. Xu. “A state-of-the-art survey of cloud manufacturing.” In: *International Journal of Computer Integrated Manufacturing* 28.3 (2015), pp. 239–250 (cit. on pp. 20, 21).
- [IJ13] J. Imtiaz, J. Jasperneite. “Scalability of OPC-UA down to the chip level enables “Internet of Things”.” In: *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*. IEEE. 2013, pp. 500–505 (cit. on p. 23).
- [JBM+17] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir, T. Eschert. “Industrial internet of things and cyber manufacturing systems.” In: *Industrial Internet of Things*. Springer, 2017, pp. 3–19 (cit. on p. 31).
- [JG17] A. Jain, M. Gupta. “Evolution and Adoption of programming languages.” In: *Evolution* 5.1 (2017) (cit. on p. 36).
- [KFLV16] F. Kretschmer, S. Friedl, A. Lechler, A. Verl. “Communication extension for cloud-based machine control of simulated robot processes.” In: *Industrial Technology (ICIT), 2016 IEEE International Conference on*. IEEE. 2016, pp. 54–58 (cit. on p. 32).
- [Kle14] M. Kleinemeier. “Von der automatisierungspyramide zu unternehmenssteuerungsnetzwerken.” In: *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Springer, 2014, pp. 571–579 (cit. on pp. 14, 20, 21).
- [LDZ18] S. Li, L. Da Xu, S. Zhao. “5G internet of things: A survey.” In: *Journal of Industrial Information Integration* (2018) (cit. on p. 32).

- [Ley11] F. Leymann. “Cloud computing.” In: *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik* 53.4 (2011), pp. 163–164 (cit. on pp. 20, 58).
- [LHL+17] Z. Luo, S. Hong, R. Lu, Y. Li, X. Zhang, J. Kim, T. Park, M. Zheng, W. Liang. “OPC UA-Based Smart Manufacturing: System Architecture, Implementation, and Execution.” In: *Enterprise Systems (ES), 2017 5th International Conference on*. IEEE. 2017, pp. 281–286 (cit. on pp. 32, 33).
- [LS18] M. P. Lewis, C. Shaw. *System and method to facilitate real-time communications and content sharing among users over a network*. US Patent 9,893,908. Feb. 2018 (cit. on p. 32).
- [LYZ+17] T. Y. Lin, C. Yang, C. Zhuang, Y. Xiao, F. Tao, G. Shi, C. Geng. “Multi-centric management and optimized allocation of manufacturing resource and capability in cloud manufacturing system.” In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 231.12 (2017), pp. 2159–2172 (cit. on p. 31).
- [LZW+10] B.-H. Li, L. Zhang, S.-L. Wang, F. Tao, J. Cao, X. Jiang, X. Song, X. Chai. “Cloud manufacturing: a new service-oriented networked manufacturing model.” In: *Computer integrated manufacturing systems* 16.1 (2010), pp. 1–7 (cit. on p. 13).
- [Man11] P. Manenti. “Building the global cars of the future.” In: *Managing Automation* 26.1 (2011), pp. 8–14 (cit. on p. 13).
- [MBM+12] F. Maciá Pérez, J. V. Berna-Martinez, D. Marcos-Jorquera, I. Lorenzo Fonseca, A. Ferrándiz Colmeiro, et al. “Cloud agile manufacturing.” In: (2012) (cit. on p. 20).
- [MBS+12] M. Melik-Merkumians, T. Baier, M. Steinegger, W. Lepuschitz, I. Hegny, A. Zoitl. “Towards OPC UA as portable SOA middleware between control software and external added value applications.” In: *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE. 2012, pp. 1–8 (cit. on p. 32).
- [MG+11] P. Mell, T. Grance, et al. “The NIST definition of cloud computing.” In: (2011) (cit. on pp. 19, 20).
- [MWB17] M. Müller, E. Wings, L. Bergmann. “Developing open source cyber-physical systems for service-oriented architectures using OPC UA.” In: *Industrial Informatics (INDIN), 2017 IEEE 15th International Conference on*. IEEE. 2017, pp. 83–88 (cit. on p. 32).
- [NBK+15] O. Niggemann, G. Biswas, J. S. Kinnebrew, H. Khorasgani, S. Volgmann, A. Bunte. “Data-Driven Monitoring of Cyber-Physical Systems Leveraging on Big Data and the Internet-of-Things for Diagnosis and Control.” In: *DX@ Safeprocess*. 2015, pp. 185–192 (cit. on p. 13).
- [Obj17] Object Management Group. *The Unified Modeling Language Specification*. Ed. by Object Management Group. Version Version 2.5.1. Dec. 5, 2017. URL: <https://www.omg.org/spec/UML/2.5.1/PDF> (cit. on pp. 17, 18, 21, 42, 49–51).
- [OPC17a] OPC Foundation. *OPC Unified Architecture Specification Part 1: Overview and Concepts*. Ed. by O. Foundation. Nov. 22, 2017. URL: <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/> (cit. on pp. 23–25, 54).

- [OPC17b] OPC Foundation. *OPC Unified Architecture Specification Part 4: Services*. Ed. by O. Foundation. Nov. 22, 2017. URL: <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/> (cit. on p. 24).
- [OPC17c] OPC Foundation. *OPC Unified Architecture Specification Part 5: Information Model*. Ed. by OPC Foundation. Nov. 22, 2017. URL: <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model/> (cit. on p. 54).
- [OPC18] OPC Foundation. *OPC Unified Architecture Specification Part 3: Address Space Model*. Ed. by O. Foundation. Apr. 4, 2018. URL: <https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model/> (cit. on p. 23).
- [PHMW18] F. Peters, I. Hartl, C. Mohr, L. Winkelmann. “OPC UA to DOOCS Bridge: A Tool for Automated Integration of Industrial Devices Into the Accelerator Control Systems at FLASH and European XFEL.” In: (2018) (cit. on p. 32).
- [RZW+17] L. Ren, L. Zhang, L. Wang, F. Tao, X. Chai. “Cloud manufacturing: key characteristics and applications.” In: *International Journal of Computer Integrated Manufacturing* 30.6 (2017), pp. 501–515 (cit. on pp. 31, 32).
- [SKS+17] J. Schlechtendahl, F. Kretschmer, Z. Sang, A. Lechler, X. Xu. “Extended study of network capability for cloud based control systems.” In: *Robotics and Computer-Integrated Manufacturing* 43 (2017), pp. 89–95 (cit. on p. 32).
- [SSKD11] T. Sauter, S. Soucek, W. Kastner, D. Dietrich. “The evolution of factory and building automation.” In: *IEEE Industrial Electronics Magazine* 5.3 (2011), pp. 35–48 (cit. on p. 13).
- [TCC+17] F. Tao, J. Cheng, Y. Cheng, S. Gu, T. Zheng, H. Yang. “SDMSim: a manufacturing service supply–demand matching simulator under cloud environment.” In: *Robotics and computer-integrated manufacturing* 45 (2017), pp. 34–46 (cit. on p. 31).
- [TCD+14] F. Tao, Y. Cheng, L. Da Xu, L. Zhang, B. H. Li. “CCIoT-CMfg: cloud computing and internet of things-based cloud manufacturing service system.” In: *IEEE Transactions on Industrial Informatics* 10.2 (2014), pp. 1435–1442 (cit. on p. 21).
- [THZ10] F. Tao, Y. Hu, L. Zhang. “Theory and practice: optimal resource service allocation in manufacturing grid.” In: *Beijing: ChinaMachinePress* (2010) (cit. on p. 13).
- [TKG+17] G. Tuna, D. G. Kogias, V. C. Gungor, C. Gezer, E. Taşkın, E. Ayday. “A survey on information security threats and solutions for machine to machine (M2M) communications.” In: *Journal of Parallel and Distributed Computing* 109 (2017), pp. 142–154 (cit. on p. 31).
- [TMTR15] S. Tedeschi, J. Mehnen, N. Tapoglou, R. Rajkumar. “Security aspects in Cloud based condition monitoring of machine tools.” In: *Procedia CIRP* 38 (2015), pp. 47–52 (cit. on p. 21).
- [TSSJ17] T. Tanaka, M. Skoglund, H. Sandberg, K. H. Johansson. “Directed information and privacy loss in cloud-based control.” In: *American Control Conference (ACC), 2017*. IEEE. 2017, pp. 1666–1672 (cit. on p. 32).
- [TV07] A. S. Tanenbaum, M. Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007 (cit. on p. 24).

- [VOX+05] S. Venkatesh, D. Odendahl, X. Xu, J. Michaloski, F. Proctor, T. Kramer. “Validating portability of STEP-NC tool center programming.” In: *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers. 2005, pp. 285–290 (cit. on pp. 13, 23).
- [WGRS13] D. Wu, M. J. Greer, D. W. Rosen, D. Schaefer. “Cloud manufacturing: Strategic vision and state-of-the-art.” In: *Journal of Manufacturing Systems* 32.4 (2013), pp. 564–579 (cit. on pp. 14, 20, 21).
- [WSJ17] M. Wollschlaeger, T. Sauter, J. Jasperneite. “The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0.” In: *IEEE Industrial Electronics Magazine* 11.1 (2017), pp. 17–27 (cit. on pp. 13, 31).
- [Xu12] X. Xu. “From cloud computing to cloud manufacturing.” In: *Robotics and computer-integrated manufacturing* 28.1 (2012), pp. 75–86 (cit. on pp. 13, 14, 20–22).
- [ZKRC17] M. Zhao, A. Kumar, T. Ristaniemi, P. H. J. Chong. “Machine-to-Machine Communication and Research Challenges: A Survey.” In: *Wireless Personal Communications* 97.3 (2017), pp. 3569–3585 (cit. on p. 32).
- [ZZZ+18] L. Zhou, L. Zhang, C. Zhao, Y. Laili, L. Xu. “Diverse task scheduling for individualized requirements in cloud manufacturing.” In: *Enterprise Information Systems* 12.3 (2018), pp. 300–318 (cit. on p. 31).

All links were last followed on May 7, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature