Cloud Robotics for 5G

Jacob Mejvik

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6020
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2016

Abstract

Technological advances in cloud technology, robotics, and telecommunications could provide for new and interesting applications. This thesis investigates how a combination of these areas can be utilized in order to improve performance in a smart manufacturing setting. In doing so, an architecture based on the Actor model has been used. The proposed architecture is both described from a theoretical point of view and implemented as a proof of concept. In the implementation the ABB robot Yumi is used to solve a 2x2 Rubik's cube with a solver executing in a cloud environment. Furthermore, the implementation has been designed to fit with future generations of telecommunications, e.g., 5G. In order to achieve this a framework developed by Ericsson AB, called Calvin, has been used. The results show that the proposed architecture can be implemented as well as point out areas for future research or development. In particular, dynamic aspects were found to be interesting areas for future work. In addition, it was concluded that business aspects are bound to play a vital role in the development and use of these architectures. Therefore, some scenarios, showing how the technical benefits of the proposed architecture can be leveraged to add economic value, are presented.

Acknowledgements

I would like to express my gratitude to my supervisors Anders Robertsson and Johan Eker for their support during the work with this thesis. I especially appreciate their willingness to help out, even outside of normal business hours and during their own vacations. Furthermore, I would like to thank the Calvin team at Ericsson for their quick replies to whatever queries I have had. In particular, Ola Angelsmark, Harald Gustafsson, and Robert Marklund deserve a special thanks. In addition I would like to thank Maj Stenmark for her patience with me in the robot lab and many practical tips on handling Yumi. Finally, all the members of the Department of Automatic Control deserve a special mentioning for cheering me on.

Contents

1

Introduction

This document is the final report of a Master's thesis project at the Department of Automatic Control at Lund University in 2016. The project is a joint project between Ericsson AB and the Department of Automatic Control. The report is structured to first give a background to the problem domain in Chapter 1. Chapter 2 gives a theoretical background. In Chapter 3 follows a practical background. Chapter 4 discusses implementation details. Results are presented in Chapter 5, leading into a discussion in Chapter 6 and conclusion in Chapter 7. Finally, Chapter 8 presents areas for future research.

1.1 Background

In the last few decades the technological advances in telecommunication, processing power, and software have provided many new possibilities. Some of these possibilities have resulted in new products and services, such as smartphones, which are now taken for granted as a part of everyday life. In keeping with this development it is believed that many more physical devices will be connected to the internet in the future. This notion forms the basis for the concept of the Internet of Things (IoT) which describes a situation where everyday objects are connected to the internet [Mattern and Floerkemeier, 2010]. In addition to being a new and innovative technology, the IoT industry potentially could generate up to $11 trillion a year in economic value by 2025 [Manyika et al., 2015]. What is more, about 30 % of this value is thought to be generated in factories. Hence, the possibilities for further improvement in the industry are considered vast and there are a number of initiatives to exploit these. However, the nomenclature is somewhat confusing with several terms that overlap. For instance, in Germany the term Industrie 4.0, referring to a fourth industrial revolution, has been coined whereas the concept is also referred to as smart manufacturing, cyber physical systems (CPS) and Industrial IoT (IIoT) to name a few [Davies, 2015]. The idea in all these cases is to create an industry where devices communicate with each other and provide feedback in order to improve the performance of the entire plant in terms of throughput, flexibility, and robustness.

However, the adoption rate in industry is hampered by difficulties in evaluating the investments as profitable [Zühlke, 2016]. To further improve the understanding of these challenges this thesis investigates how robotics can be combined with cloud technology and telecommunications to facilitate advances in industry. In doing so it has been deemed that the term smart manufacturing fits best with the content and will henceforth be used.

Robotics

The word "robot" has an interesting history as it was first created as a fictional description of humanoids in a 1920's play by Czech writer Karel Čapek and has since been further elaborated on in fiction [Cook, 2015]. Although there are similarities, the fictional and scientific views on robots differ and clarification of the term is required. Furthermore, this thesis will mainly be concerned with a specific type of robots, namely industrial robots, which are defined in ISO 8373 as:

> An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications.
>
> [IFR, 2015]

This definition does, however, raise some further questions about the terminology and ISO 8373 gives the following clarifications:

> **Reprogrammable**: whose programmed motions or auxiliary functions may be changed without physical alterations;
>
> **Multipurpose**: capable of being adapted to a different application with physical alterations;
>
> **Physical alterations**: alteration of the mechanical structure or control system except for changes of programming cassettes, ROMs, etc.
>
> **Axis**: direction used to specify the robot motion in a linear or rotary mode

In addition, being a manipulator implies that a robot is able to alter other objects. An example of such an industrial robot is shown in Figure 1.1.

Figure 1.1 ABB's IRB 2400 robot. Photo:ABB

The use of robots in the industry is gradually increasing all over the world [IFR, 2012]. In addition to defining what is meant by the term robot, it is also useful to consider when robots are beneficial to use. Considering a generic product, one can conclude that its production can be divided into one or several tasks, such as assembly or painting, that have to be performed. Deciding on how to perform these tasks is often a matter of deciding the level of automation to be used. At one extreme the task is performed manually and at the other extreme specialized machinery is used. Deciding on the level of automation is often connected to the length of the production series. If a large series is to be produced it is likely that it is worth investing in specialized machinery, whereas if the series is short it is likely that a more manual approach is favored. Robots, being reprogrammable, multipurpose manipulators, on the other hand provide a combination of repeatability and flexibility. They are typically less efficient than specialized machinery at performing a specific task, however, they are flexible and can be used for different tasks. Compared to the manual approach they perform tasks with a higher degree of precision and repeatability but require setup and programming before production can commence. Therefore, robots are favored somewhere in between the two extremes, as seen in Figure 1.2 [Nikoleris, 2016]. Furthermore, with a trend towards mass customization a more flexible mode of production is favored [Sheng et al., 2009].

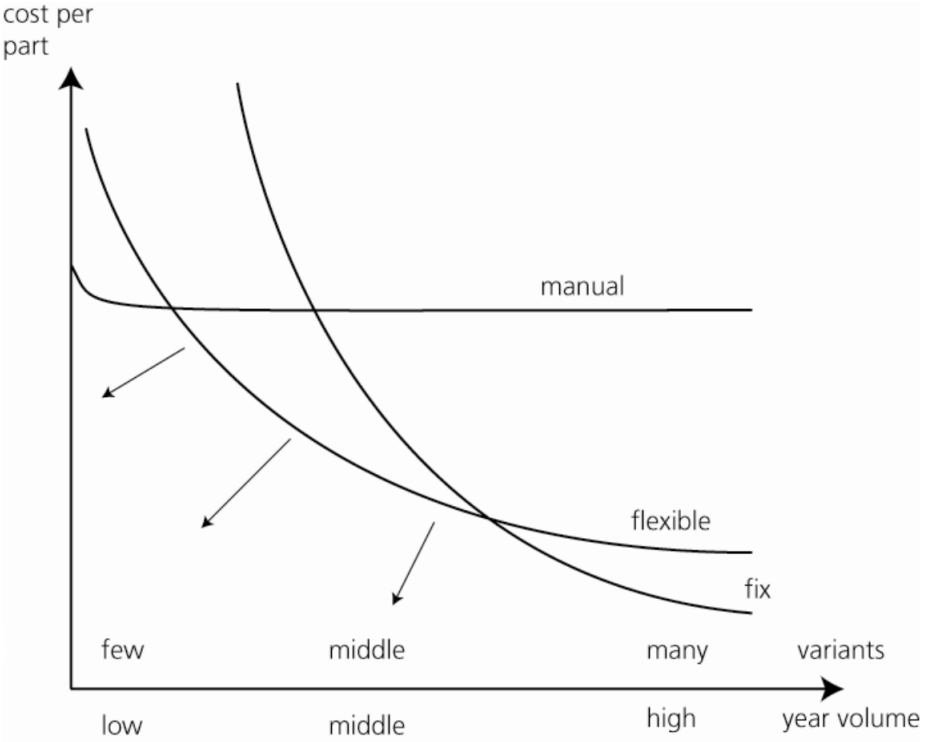

Figure 1.2 Cost per part as a function of production volume for different levels of automation

Cloud technology

Another issue connected to smart manufacturing is how to make use of all the information accumulated from the vast array of IoT devices, and combine it with other production data. Some of the data has real-time constraints and the latency can therefore not be too high. In addition there is the issue of avoiding costly downtime in the plant due to malfunctioning equipment or software upgrades [Brandl, 2016]. From this reasoning it follows that it is desirable to keep the complexity of the machinery as low as possible, thereby facilitating efficient replacement of faulty parts. An interesting approach to these challenges is to make use of cloud technology. In a cloud setup, similarly to a conventional data center, computational power is located at one specific physical location. However, when using cloud technology the resources at this physical location are not dedicated and are instead allocated on demand. By taking this approach the maintenance costs are lowered since it is easier to do efficient maintenance when all equipment is located in the same place [Angeles, 2013]. Furthermore, the utilization of the resources can be increased since the peak

demand for computational power is unlikely to occur simultaneously. Whereas the utilization rate in a conventional data center is in the range of 12-18 % hyper-scale cloud providers could reach 40-70 % utilization [Delforge and Whitney, 2014].

Telecommunications - 5G

In recent years there have been significant developments in the capacity of wireless communication resulting in an increasing demand for these services. The increasing demand shows no signs of declining and according to the latest mobility report by Ericsson the accumulated data traffic in the coming five years will be 12.5 times greater than it has been in the previous five years [Cerwall et al., 2016]. Hence, it is not surprising that significant efforts are put into developing the capacity even further. Although the current generation,4G/LTE, has not been fully deployed across the globe yet the telecommunications industry is already planning for the next generation, 5G, to start deployment around 2020 [GSA, 2015]. However, exactly what 5G entails is not entirely clear as the standard has not yet been defined. There are many considerations to be taken into account when defining the standard, such as data rate, latency, energy usage and cost [Andrews et al., 2014]. Nonetheless, the improvements will likely make new products and services feasible.

When it comes to smart manufacturing and process control, telecommunication plays a vital role by providing a spatial flexibility both in terms of mobility and in terms of where sensors can be placed. However, the current latency and data rates impose constraints on the domain of possible applications. Particularly real-time applications, which are sensitive to delay, suffer from these constrains. For these applications improved telecommunications could allow for better use of sensor data as well as allow designs where the controller is not physically in close proximity to the process. Hence, from an industrial point of view 5G has the potential to push the boundaries of what is possible further.

Combining robotics, cloud technology and 5G

Given the advances in robotics, cloud technology and 5G it is interesting to see what can be achieved if these areas are combined. One of the limitations of using robotics is that new instructions have to be provided to the robot relatively frequently in order to make use of its flexibility. Furthermore, to utilize the flexibility fully the robot may have to move to a new location to perform the next task, making it impossible to use fixed communication channels. Telecommunications, 5G in particular, could provide the solution to this problem by enabling lower latency and less jitter compared to previous generations. Another issue that has been discussed above is to make use of all the sensor data available. Some of this data may be collected by sensors on the robot whereas other data is collected from sensors that are placed remotely, perhaps in another robot, in the operator's cellphone or as a fixed installation in the factory. An example of this would be camera images used to calculate coordinated trajectories for robots or to identify work objects. Using

telecommunications will in this case enable a flexible transfer of data and allow for easy replacement. However, since many of these calculations are computationally heavy the problem of performing the calculation remains. In some cases it is likely that the computational resources available locally are insufficient and a solution with cloud technology would be preferable. Although there are clear benefits there are also several issues that need further investigation. For example, one consideration would be where calculations should be performed in a distributed architecture. Since telecommunications imposes restrictions in terms of latency and throughput it is not always certain that it is optimal to perform all calculations in the cloud despite the superior computational power and economic efficiency.

1.2 Purpose

As described above, using cloud technology and telecommunications for robotics seems very promising. However, exactly how this should be done needs further research. Hence, the purpose of this thesis is to investigate an architecture for robotics that uses cloud technology and can be extended to fit future generations of telecommunications (5G).

1.3 Problem

In order to further clarify the problem at hand the following three questions will serve as the problem statement.

1. What is a suitable software architecture for cloud robotics using 5G?
2. What is a suitable controller architecture for cloud robotics using 5G?
3. What requirements does cloud robotics impose on 5G?

Furthermore, the proposed architecture will immediately pose the question of whether or not it is possible to implement in practice. Hence, the problem formulation also extends to providing a demo that serves as a proof of concept. A successful demo should showcase all the critical features of the proposed architecture and allow for extensions through easy modifications. Likewise, the business minded reader will instantly reflect on the commercial merits of the suggested architecture. Although this thesis has no intention to give a full coverage of the business aspects, some discussion is warranted in connection to opportunities that could arise from the proposed architecture.

1.4 Delimitations

The main aim of this thesis is not to provide a comprehensive solution to all issues concerning the problem at hand, but rather to implement a demo that works as a proof of concept as well as to point out areas for future work. Given this aim it is not always necessary to provide full coverage of all possible alternate solutions and sometimes it is motivated to use a solution known to work rather than the ideal solution. Furthermore, since the project is of an exploratory nature the thesis work will be adapted to fit the given time frame. In addition, the work will be constrained to using existing software that in some cases is under development which will impose more constraints on what can be achieved in the given time frame. From a practical point of view this implies that if it is deemed unfeasible to develop desired features during the thesis work, these features will be described in the report rather than being implemented in the demo.

1.5 Method

The method used to address the problems can be divided into two parts. The first part is comprised of a literature study and an investigation of available software that could be of interest. The second part is the actual implementation of the architecture. From a practical point of view a crude architecture was created in the preliminary stages prior to the thesis work had actually started and when only a limited literature study had been performed. This crude design pointed out a number of implementation issues that have to be addressed regardless of the choice of architecture. Hence, the work proceeds with the implementation issues and the literature study in parallel. Due to the exploratory nature of the project, unforeseen issues are bound to appear and the strategy for handling these is to prioritize hard on what actually is necessary for the demo while simultaneously providing an approach that can be used for the general case. The final result will be reached through an iterative approach where prioritization is done gradually, thereby making it possible to conclude the work when appropriate to meet the deadline. This approach is reminiscent of the approaches used for agile software development [Ionel, 2009]. The work will be considered successful if a working demo has been implemented and an architecture complying with the demands of the problem domain has been found.

2

Theoretical background

This chapter describes central concepts to the thesis at hand from a theoretical point of view. As the problem domain touches on several large research areas some of the concepts are only described briefly as to inform the reader about key concepts and point out where additional material can be found.

2.1 Actor model

First introduced in 1973 by Hewitt *et al* within the artificial intelligence domain, the Actor model is a programming model designed for situations with a high degree of parallelism [Hewitt et al., 1973]. Hewitt and his PhD students have since contributed to further expansions and formalization [Hewitt, 2007][Agha, 1986]. The model forms the basis for concurrency handling in several computational languages, such as Erlang and Scala [Karmani et al., 2009][Stivan et al., 2015]. Furthermore, the model has been used to implement pure actor-based languages, such as the CAL language developed within the Ptolemy II project at UC Berkley [Eker and Janneck, 2003]. More recently, the model has received attention as multi-core processing has become the standard [Charousset et al., 2013].

Actors

As Hewitt *et al* (1973) describes in their paper an actor is an isolated entity which operates by declaring its intentions, i.e., what it promises to do, and fulfilling them [Hewitt et al., 1973]. This setup is akin to a contractual situation where the actor promises to deliver a specific result given certain inputs. The internal actions within the actors are called events and consist of an incoming message that results in one or several outgoing messages [Hewitt and Baker, 2002]. Within these events the actor behavior is enclosed. Furthermore, Hewitt and Baker consider system of actors where several actors are connected. In this thesis, such a system of actors will later be referred to as applications.

Asynchronous message passing

Due to the construction of actors as isolated entities without knowledge of the state of other actors it follows that no common time exists. Besides, as Agha explains, as the actors are isolated entities and communication between the actors is outside the control of any actor, there is no guarantee that a common time could be maintained even if initially synchronized [Agha, 1986]. Alas, the actor model is inherently asynchronous and embraces this by explicitly stating that the message passing is asynchronous. In practice this means that there are neither any guarantees nor requirements on the timing aspects. Messages that are sent can be received in another order and the system should still work. However, within an actor it is guaranteed that received messages are ordered by imposing an arbitration principle that enforces ordering [Hewitt and Baker, 2002].

2.2 Robotics

A general introduction to robotics was given in Section 1.1 and this section elaborates further on important aspects.

Robot kinematics

As stressed in the definition of a robot given in Section 1.1 an important feature of a robot is that it should be automatically controlled. In order to achieve this a special branch of mechanics concerned with the motion of bodies, namely kinematics, is used [Corke, 2011, p. 137]. In doing so, the robot is often represented schematically so that all the joints and links can be identified, see Figure 2.1. Each joint can be of one of two types, either revolute (rotational) or prismatic (sliding). Whereas the links are fixed the joints can be altered individually. By altering the joints the position of the robot is changed and hence it is said that each joint provides one degree of freedom [Corke, 2011, p. 137]. Controlling the position of the robot is therefore a matter of finding the appropriate values for the joints which in turn is a problem of combining several transformation matrices and solving a system of equations. One way of doing this is to use the Denavit-Hartenberg convention which provides a systematic labeling of the links and joints [Corke, 2011, p. 138]. Although there are several issues to account for when solving this system of equations, such as singularities, it is still not sufficient to control the robot since the dynamic aspects are neglected. Hence, in order to account for dynamic aspects such as velocity and acceleration in the robot arm, a more elaborate representation has to be made. Since the focus of this thesis is not on the detailed control mechanisms in the robot only a brief account will be given.

By studying the problem and applying some algebra the system of differential equations describing the motion of the robot can be found [Corke, 2011, p. 191]. The details are somewhat complicated but when these are ironed out one ends up with a system of differential equations shown in Equation 2.1

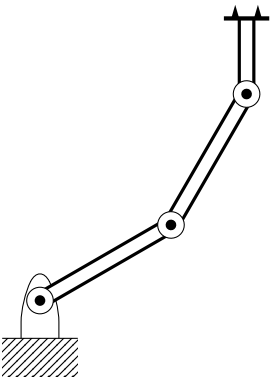

Figure 2.1 Schematic representation of a robot with three joints

$$Q = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + J(q)^T g \tag{2.1}$$

The system of differential equations is in matrix notation where q, $\dot{q}$ and $\ddot{q}$ are vectors with joint coordinates, velocities and accelerations, respectively. M is the joint-space inertia matrix, C is the Coriolis and centripetal coupling matrix, F is the friction force, G is gravity and J is the manipulator Jacobian. Given the pose, velocity and acceleration Equation 2.1 calculates the forces and torques, Q, required at each joint in order to reach the desired position.

Robot controller

In order to control the position of the joints, a number of cascade controllers are used. The controller shown in Figure 2.2 is an example of such a controller. The structure is a cascade controller with position and velocity references, combined with a torque feed forward.

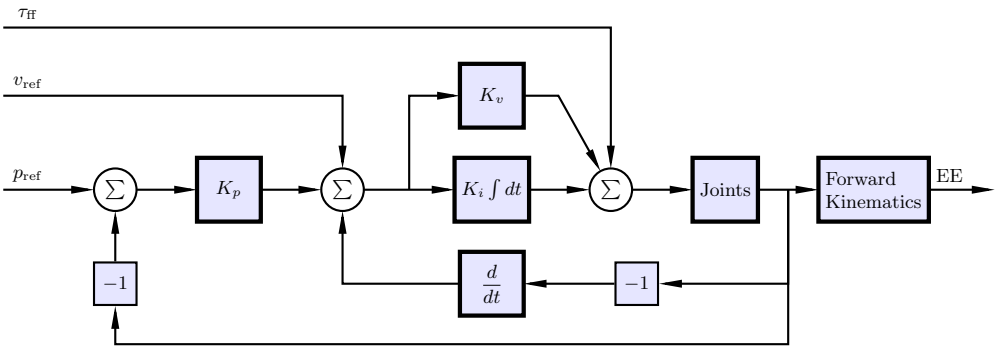

Figure 2.2 Schematic robot joint controller. [Department of Automatic Control LTH, 2016]

Safety

One of the major concerns regarding robots is the interaction with humans in their vicinity. If no safety measures are in place and a robot is instructed to perform a certain movement, it will try to perform that movement regardless of obstacles in its way. Since robots are typically made from hard materials and humans are soft there is a large risk for injuries. The solution to this problem is of course to use safety measures and for traditional industrial robots safety cages or zones are used [Corke, 2011, p. 2]. However this approach is somewhat impractical especially considering that some tasks have to be performed close to humans. A clear example of this is medical robots to help with heavy lifting of patients. In an industrial environment the same problems exist as some tasks are suitable for robots and some for humans causing humans and robots to operate close to each other.

2.3 Wireless

There is a number of different wireless protocols that could be used for IoT [LinkLabs, 2016]. Though, this thesis has no intention for accounting for them in detail, there are a few common features that are important to be aware of.

Licensed and unlicensed spectrum

One of the main design considerations for wireless networks is which spectrum to use. Since the medium the signals travel through is the air it is easily accessible to anyone. What's more, the available spectrum is limited and hence is a valuable commodity [Bazelon, 2009]. In order to ensure some sort of order in this easily accessed medium a division between licensed and unlicensed spectrum is done. The unlicensed spectrum is free for anyone to use at any time, whereas the licensed part requires permission [Anthony, 2012]. Licenses for this part are usually sold by

the government and last for a pre-specified time. By limiting the number of agents that have a right to use a specific spectrum it is easier to find a common standard and thereby reducing the interference.

Random Access

Yet another design issue is how the channel is accessed. In a rudimentary scheme, when a node has data to send it will attempt to send it. This might result in collisions which in turn will lower the total throughput in the network [Kleinrock and Tobagi, 1975]. For a network where all the nodes and their properties are known in advance it might be possible to construct a schedule-based protocol. However this approach scales very poorly and hence, a scheme that can handle more uncertainty is preferred. Such schemes are called random access schemes and are decentralized schemes based on the notion that nodes will have data to send at irregular intervals. As pointed out above the randomness results in a higher risk for collisions and to address this issue random access schemes are designed with a number of features, such as back-off and Quality-of-Service settings, limiting the negative effects. For IoT the scaling aspects are important as the number of devices is predicted to be large and a number of variations of random access schemes have been investigated [Hasan et al., 2013]. Nonetheless, all these schemes are intrinsically random and there are no guarantees that a transmission will be completed in a given time period.

4G architecture

Accounting for the 4G architecture in detail is outside the scope of this thesis. However, in order to provide some fundamental knowledge Figure 2.3 shows a schematic of the architecture. As can be seen in the figure there are several nodes and acronyms. Most of these will not be described and for a full description the reader is referred to other sources. For the purpose of this thesis the most important nodes are the eNodeB, which is the radio access point, and the user equipment (UE) [Abed et al., 2012]. In an IoT setting the UE would be an IoT device.

The next generation - 5G

Exactly what is meant by 5G is at the time of writing unclear as no formal specification has been published. There are however a few requirements that have been identified, namely an increased data rate, lower latency, decreased costs and decreased energy consumption [Andrews et al., 2014]. Although the precise demands on each of these requirements have not been finalized they point in a general direction of increased performance. In terms of the infrastructure to achieve this it will in this thesis be assumed that it will be similar to 4G. Hence, it is inferred that there will be a node akin to the eNodeB in 4G connected to a wired net and communicating with the mobile units through a wireless channel. Given this setup the eNodeB equivalence in 5G will be the lowest common level for communication between

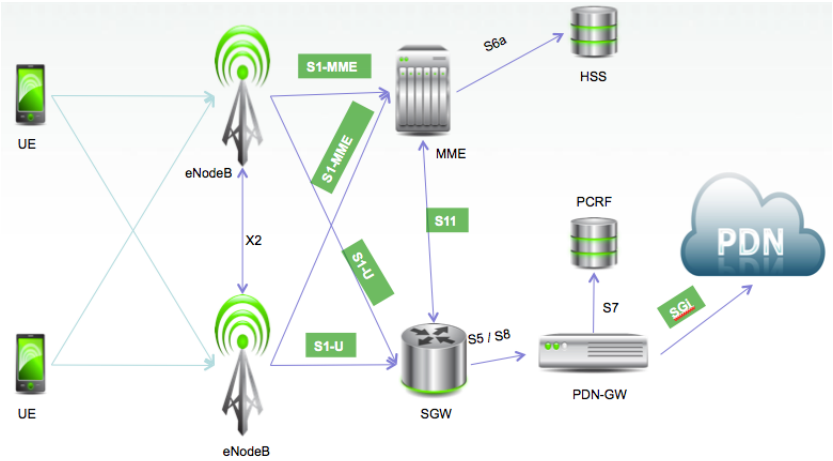

Figure 2.3 4G architecture. [Ixia, 2011]

mobile units. Hence, the eNodeB equivalence is an interesting point to perform calculations on data collected or shared by several mobile units. Moreover, it will be assumed that 5G will be built using Network Functions Virtualization (NFV). NFV is intended to address the difficulties with a heterogeneous hardware configuration that is currently the standard in the telecommunications industry. By virtualizing the functions of the hardware it can be run on standardized platforms placed either in centralized data centers or spread out in the architecture [Chiosi et al., 2012]. This would mean that standardized equipment would be available throughout the architecture which in turn means a greater flexibility in terms of running software in a distributed fashion.

2.4 Cloud

As previously discussed in Section 1.1 the main driving force behind using cloud technology is to increase the utilization of hardware and thereby reduce the cost associated with IT infrastructure. The meaning of the cloud from a more technical aspect has not yet been addressed and this section will discuss it. The word "cloud" in itself suggests a somewhat unclear meaning of the term. In fact there is an ambiguity in what a cloud service would provide and in an attempt to address this issue cloud services are divided into three categories. The first category is called Infrastructure as a Service (IaaS). IaaS is the category closest to the hardware. Essentially this means that very little infrastructure is provided but also that the user gets a direct access to many of the resources. The second category is a Platform as a Service (PaaS) which means that a programming or execution environment is provided. As compared to IaaS the user would be able to create or run programs

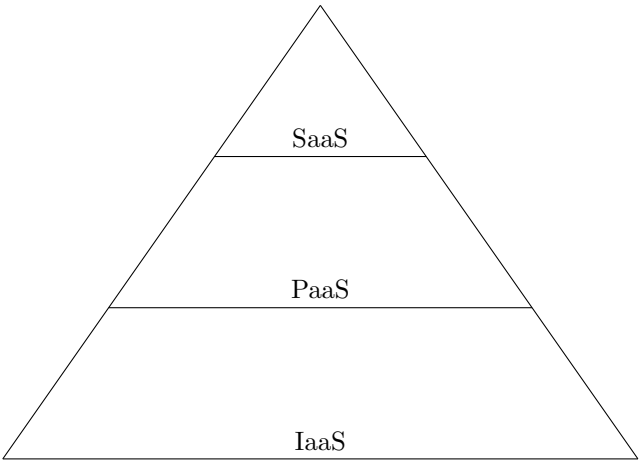

Figure 2.4 Cloud stack

without addressing the details of setting up these environments. The third category is Software as a Service (SaaS) where an entire application is provided to the user, for example Google Docs. Depending on the type of application, developers would choose to work either on a PaaS or IaaS level in order to provide end customers with an application on the SaaS level [Lenk et al., 2009]. These three categories could be divided even further and together they comprise the so called cloud stack shown in Figure 2.4.

2.5 Dead-time compensation

Using telecommunications typically introduces delay, also known as dead-time, in the control loops. Although, this aspect is not specifically covered in this thesis it is worth mentioning that some dead-time can be compensated for in the controller architecture. Hence, studying the possibilities for dead-time compensation could push the boundary for what is possible to achieve further. Exactly how the compensation should be designed depends on the particular setup and has been covered at length in various books [Normey-Rico, 2007].

2.6 Strategy

As a means of putting the proposed architecture in context a few basic business strategy concepts are used. Since the focus of this report is technical the business concepts will only be mentioned briefly and the presentation is by no means conclusive.

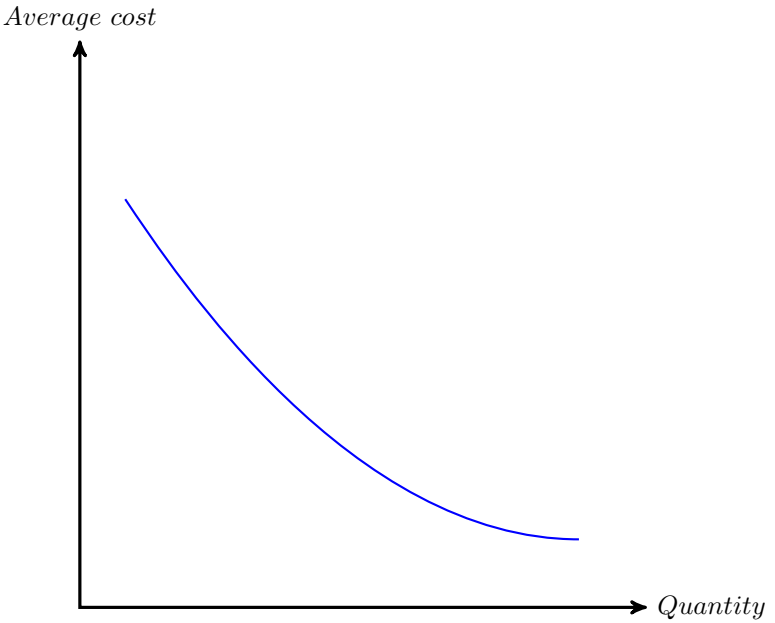

Figure 2.5 Economics of scale

Economics of scale and scope

Economics of scale refers to the intuitive notion that the marginal costs of producing a product diminishes as the production volume increases [Besanko et al., 2007, p. 114] . The reason for this is simply that the fixed costs can be spread out over a larger number of products. As shown in Figure 2.5 the rate at which the average cost is lowered becomes slower as the quantity increases. Economics of scope is a similar, but more debatable, concept alluding to that producing a wider range of products in an analogous way would push the marginal costs downwards [Besanko et al., 2007, p. 53]. Economics of scale and scope have been used to explain much of the productivity development in 20th century industry [Teece, 1993].

Competitive advantage

The concept of competitive advantages gives an explanation to why firms earn money. A firm's competitive advantage essentially points to the aspects in which they perform better than their competitors. As Porter argues, a firm should carefully examine how it adds value to its customers and thereby understand what its competitive advantage is constituted of [Porter, 1991]. One way of doing this is to investigate the company using Porter's value chain. As shown in Figure 2.6 the value chain identifies key activities in a firm and splits them into primary and support ac-

tivities. By studying these activities one by one it should become clear what a firm is good at and where it adds value. Note that the value added does not necessarily stem from the primary activities.

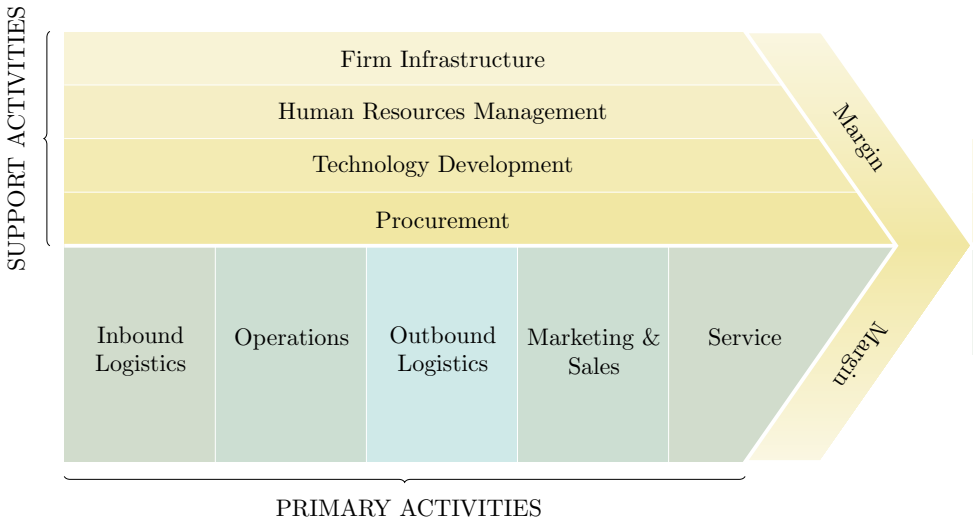

Figure 2.6 Porter's value chain

Transaction costs

In his famous article, The Nature of the Firm, future Noble Laureate Coase first introduced the concept of transaction costs as an explanation for why and when firms should purchase from the market, as opposed to producing themselves [Coase, 1937]. The concepts have henceforth been further developed by another Noble laureate, Williamsson [Williamson, 1981]. Essentially the theory suggests that although the open market normally is an efficient way to acquire assets, costs are associated with buying from the market. These costs are typically connected to contract difficulties, specificity of the asset and information asymmetries. Hence, the decision to buy or make is determined by whether the transaction costs are low enough to be outweighed by the improved efficiency of the market [Besanko et al., 2007, p. 105-135].

3

Practical background

This chapter discusses practical aspects leading up to the choice of architecture. Whereas the theoretical chapter is general in its nature, this chapter has an increased focus on aspects directly relevant to the implementation at hand. Hence, the starting point is a basic architecture, which is described in Figure 3.1. As seen in the figure, the basic architecture is fairly straightforward with a robot, shown as a manipulator, communicating with a cloud server over a 5G link. The 5G link is divided into two parts, a wireless part (dotted line) and a wired part (full line). However, dwelling deeper into the architecture one soon realizes that there are a number of issues that have to be taken into account.

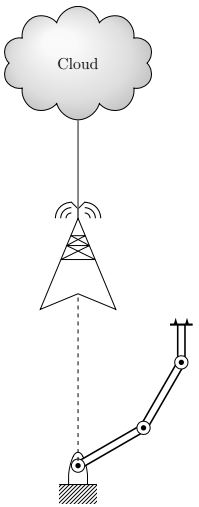

Figure 3.1 Basic architecture: A robot communicating with the cloud over 5G

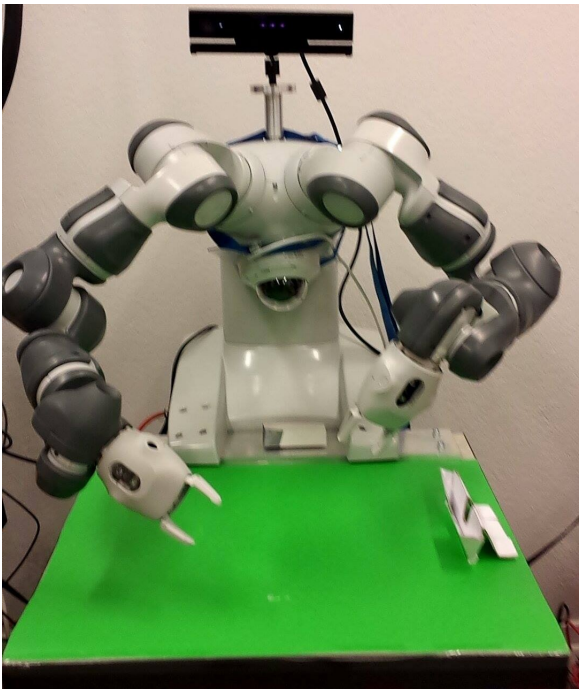

Figure 3.2 Yumi robot in the robotics lab at LTH

3.1 The Robot

Since the focus of this thesis is to investigate how current technological advances can be used in the future it is natural to also use an advanced robot. The robotics lab at LTH has access to one of ABB:s newest robots, the ABB Yumi shown in Figure 3.2. Yumi is a two-armed robot with seven joints per arm, designed to be safe for humans to work nearby. In addition, Yumi has been given features that simplify the operator work such as lead-through, which lets the operator guide the robot to a position by physically moving the robot arms [ABB, 2015a]. Providing Yumi with instructions is primarily done using ABB:s robot programming language RAPID. The RAPID code is run directly on the robot and the on-board computer calculates the control signals to the individual joints. As a final note, it should be mentioned that Yumi could be exchanged for any other industrial robot without affecting the principal architecture described here.

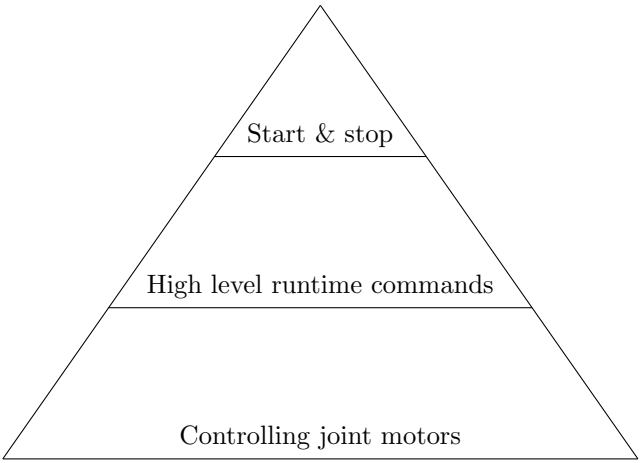

Figure 3.3 Three levels of communication

3.2 Robot communication

Since this thesis is dependent on communicating with the robot using remote resources the options for communication channels are of interest. Three distinct levels of communication have been identified. At the highest level only basic commands, such as start- and stop-execution are allowed. At this level no runtime commands can be sent. However, the entire program can be replaced when execution is stopped. At the second level targets and commands are sent during runtime to the robot but calculation of control signals to the joint motors is left entirely to the internal computer. At the third level the joint angels are directly altered allowing compensation for offsets not handled by the internal computer. In Figure 3.3 the three levels are depicted in a pyramid to indicate that the higher levels require less input but also that the lower levels provide more access to the system.

While deciding on a suitable level it is valuable to consider the available interfaces. For the highest level ABB provides a REST API which was deemed to provide sufficient support for the basic features [ABB, 2015b]. However, the runtime limitations of the REST API makes it unfit for controller applications which require commands to be sent when the program is running. The second level has no direct support from ABB but ABB:s robot programing lanugage, RAPID, allows socket communication [ABB, 2010]. This means that in theory all the RAPID commands should be possible to execute at runtime provided that the necessary interpreters are implemented. Such a solution thereby makes it possible to construct any robot instruction, that can be run in RAPID, in other environments and send the information to the robot over a socket. Just as with any other RAPID program such a program would have to pass through the on-board computer of the robot that cal-

culates control signals to the joint motors which creates some additional overhead resulting in more delay compared to the third level. The third level, directly manipulating control signals to the joint motors without passing the on-board computer, is the same level that is used in the research interfaces for other robots in the Robotics Lab at LTH. It should however be noted that, when this is used, the usual procedure is to let the on-board computer do most of the calculations of the control signals and only make minor changes to these. The reason for why one would like to make such minor changes would for example be to correct errors induced after the on-board computer has calculated the joint angels, e.g., correcting for small work piece irregularities detected by a camera or do path corrections from a seam tracking sensor. ABB provides an interface for this, called Externally Guided Motion (EGM) where the trajectories can be altered every fourth millisecond [ABB, 2015c]. However, at the start of the thesis work it was not yet provided for the Yumi Robot in the LTH robotics lab.

After consideration it was decided that the second level will provide all the features needed to make useful applications in this thesis and the main focus was therefore directed towards this option. Possibly the REST API for the first level could also be added in order to facilitate changing programs and starting and stopping execution in a more convenient fashion. The third level also provides useful features but for the time being the insecurity regarding whether there will be support for this or not was considered too risky. Furthermore, since many of the benefits of directly manipulating the control signals to the joints are dependent on real-time constraints in the order of milliseconds to be efficient it is preferable to start with less time sensitive applications.

Upon deciding to work on the second level, using socket communication, one quickly concludes that a marshaling protocol is useful. Similar issues had arisen previously at the department of Automatic Control and such a protocol, called Labcomm, had been developed. However, the protocol had never been used to perform exactly the kind of operations that was intend in this thesis. After investigating labcomm thoroughly it was noted that it had been hard to maintain the implementation for RAPID due to lack of a command line RAPID interpreter. This caused the RAPID part to be out of sync with the remainder of Labcomm. Therefore, for this thesis, marshaling will have to be handled separately. Furthermore, due to the project constraints it was decided to keep this protocol as simple as possible and leave an improved marshaling protocol as a future add-on.

3.3 Cloud setup

The cloud used in this thesis is provided by Ericsson. The servers are set up to provide an infrastructure as a service (IaaS). Hence, there is some work to do before everything is up and running. First of all, a virtual machine, known as an instance has to be created. This instance is created from an image, in this thesis a clean

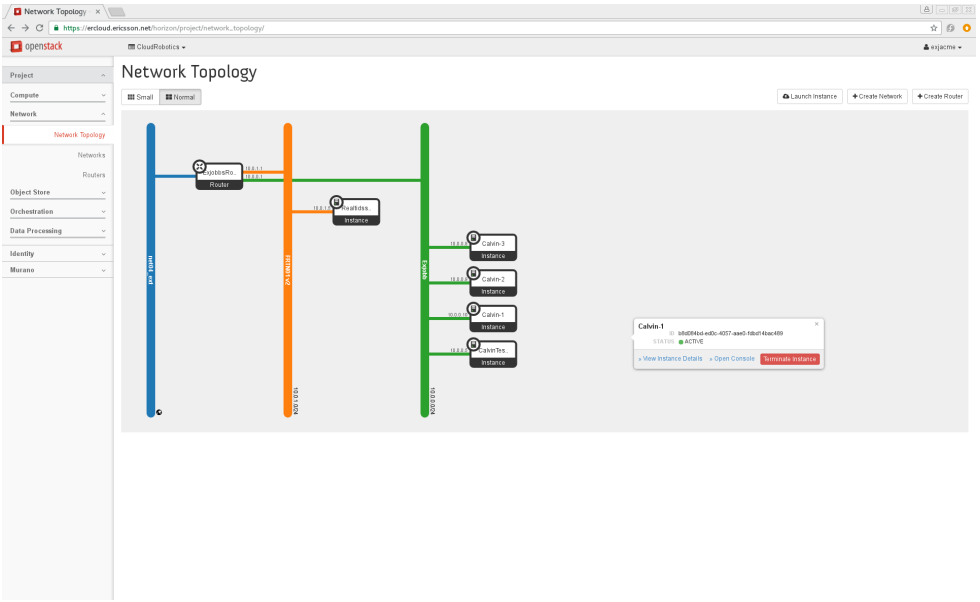

Figure 3.4 Example of network topology in cloud environment

Ubuntu distribution has been used as a base. Secondly a virtual network has to be created so that the instance can be connected to a router. Thirdly, this network has to be connected to the internet by providing it with an external IP-address. This procedure is necessary since the number of IP-addresses is limited in IPv4 and connecting all instances to the internet would quickly consume the available addresses. Finally, in order for the communication to work the firewall rules for the network have to be set up properly so that the relevant communication is allowed through on the appropriate ports. Figure 3.3 shows an example of how a network topology could look. In the figure, two networks (green and orange) are connected to the internet (blue) through a router. The networks contain one respectively four instances. The image actually shows the cloud environment on which the demo is run on. However, the exact setup is of no relevance as only one of the instances are used in the final implementation and the others have been used for test purposes.

3.4 Calvin

The main benefit with a cloud architecture is reduced costs while simultaneously providing access to increased computational power. However, in real-time constrained applications such as robotics it might, performance wise, be too costly to perform the computations remotely. This problem will be amplified when 5G is used

since it will induce uncertainty and therefore it might at times be more beneficial to perform the calculations locally or at some intermediate node. Furthermore, with increased capacity in the wireless part of the 5G infrastructure it is possible that the bottleneck will be the connection between the eNodeB equivalents and the cloud servers rather than the wireless link. Hence, there might be situations where it is of value to allow the architecture to change during runtime. It would of course be possible to construct software that handles these situations, but it would be preferable to use a framework instead to allow a more efficient reuse. There are several frameworks that could be considered for the thesis at hand but it was decided early in the work process to focus on an IoT framework devloped by Ericsson, called Calvin.

As described by the developers at Ericsson Research, "Calvin is an application environment that lets things talk to things" [*Calvin Repository* 2016]. This description highlights the flexibility of Calvin as a framework that can be used for a variety of applications and it is not specifically designed for robotics. This approach fits very well with the notion that a factory could have a multitude of sensors or other data that should be integrated with robots in order to improve production efficiency and flexibility. Calvin is an open source project and a lot of useful information can be found in the public wiki and repository online [*Calvin Repository* 2016]. Furthermore, there also exist a few papers describing the ideas behind Calvin as well as a number of Master's theses.

The following description of Calvin is based on information found in the public repository as well as in a position paper outlining the ideas behind Calvin [Persson and Angelsmark, 2015].

Basic Calvin concepts

The general motivation behind Calvin is to make it easier for developers to quickly build applications from components without worrying about too many details such as network communication and orchestration. Nonetheless it is important to know some of the basic concepts and the general architecture of Calvin.

Runtime environment At the core of Calvin is a runtime environment in which applications are run. A runtime environment in Calvin is comprised of an arbitrary number of nodes, known as Calvin runtimes, typically running on different hardware. Together these nodes form the infrastructure for Calvin and ensure that the communication within Calvin functions properly. Furthermore, each Calvin runtime has a set of capabilities that can be performed at that runtime. For example, a runtime running on a device with a thermometer connected would have the capability to measure temperature. The capabilities and the communication between the runtimes are completely disconnected from the applications making the runtimes the backbone of Calvin.

Actors Calvin is based on the Actor model and uses asynchronous message passing. This means that a Calvin application is divided into parts, known as Calvin actors, which are self-contained. Each actor has a set of incoming and outgoing

ports. When the actor receives the appropriate tokens on the incoming ports it executes and possibly sends out one or several tokens on its outgoing ports. An actor instance has no information about the state of other actor instances. Furthermore, similarly to the different capabilities of a runtime, each actor has a set of requirements for it to be able to run on a specific runtime. For example, an actor reading the temperature would have a requirement that the runtime it is running on has a temperature measuring capability.

Migration Since each actor is constructed to work in isolation and only communicates with incoming and outgoing messages these are ideally constructed to be migrated. When a migration of an actor instance, from one runtime to another, occurs the internal state is simply saved and the information is sent to the receiving runtime. At the receiving runtime the actor is reconstructed by combining the internal state and the actor blueprint. In an extended implementation it could also be possible to migrate actor instances to runtimes with no prior knowledge of the actor type by passing the blueprint of the actor as well. Migration is the feature in Calvin that allows for the high-level dynamic behavior and could offer interesting prospects in a robotics scenario. In essence, the migration service keeps track of what and when to migrate as well as executing the migration.

CalvinScript In order to specify the actual application, a CalvinScript is used. The CalvinScript specifies the actor instances that should be run and the connections between these, thereby constructing the application. In addition to the CalvinScript a deployment script can also be passed that specifies requirements on the runtime where the various application nodes are allowed to run. For example, if one imagines a runtime environment comprised of runtimes owned by different companies there might be a situation where one would like to limit the set of runtimes an actor can be deployed on based on the owner of the runtime. Similarily, another use of the deployment requirements is to assign different security classes to the runtimes and limit deployment based on that information.

Miscellaneous Calvin is intended to be a distributed framework and thereby support for distributed hashtables (DHT) has been added. By replacing the conventional registry node with a distributed registry there would be no single point of failure. Unfortunately, there are some stability issues at the time of writing so the DHT functionality has been excluded from this thesis work.

In terms of security features, these are currently being developed and enhanced. For the purpose of this thesis security issues have been disregarded but is in the more general perspective a very important area for robotics.

Benefits of Calvin for robotics

Calvin has many features that are interesting for robotics. Firstly, it provides a framework for connecting a variety of different devices and could in theory remove much of the work needed to integrate these devices. Secondly, the actor model

makes it possible to develop actors in isolation without taking other parts into account. This should also make it fairly easy to integrate Calvin with other frameworks, thereby extending the functionality rather quickly. Besides the easy integration with other frameworks it is also easy to build parts of the application outside of Calvin using socket communication if needed. Thirdly, it is easy to construct new applications from the library of existing actors and the web interface that is included makes it easy to deploy and monitor such applications. However, according to the author of this thesis, the most interesting feature from a robotics point of view is the migration functionality and the mapping of requirements to capabilities. Although, not fully implemented in Calvin yet, this functionality adds dynamic features that otherwise would be very difficult to achieve. Being able to, during runtime, change where actors are executed and thereby alter the behavior and improve the performance of robotics applications is of great interest.

Drawbacks of Calvin for robotics

Although, there are many benefits of using Calvin for robotics, there are also a number of drawbacks. The main drawback is that Calvin is still under development and lacks many of the features that hopefully will be developed in the future. Furthermore, Calvin is not specifically designed for robotics. Although this is one of the main strengths of Calvin in a more general sense, it also means that there is a lot of things that have to be handled by the developer. For example, there is no support for robotics at all, meaning that the design of all the protocols is left to the developer. There is also the issue of what should be put into Calvin, and what should be run outside. Given time, a full library of robotics actors could be created within Calvin, however this is bound to take significant time and effort. In terms of the architecture of Calvin, the asynchronous behavior creates the need for synchronization to be built into the Calvin application. This is especially important for more advanced robotic applications where collaboration between robots is of importance or when external sensors, such as cameras, are used. Similarly, real-time constrained applications could suffer from the asynchronous behavior causing difficulties in constructing reliable controllers. Furthermore, Calvin will induce some additional overhead which might be of importance for some of the more time sensitive applications.

3.5 Calvin - The big picture

As mentioned in the introduction to this chapter, the basic setup is fairly simple. However, one of the strengths of Calvin is that it is designed to work in a far more complicated setup. For example, consider the network topology shown in Figure 3.5. This topology contains several network nodes and a number of IoT-devices connected to these. Scaling up the basic setup, this is the kind of topology that is envisioned. Each of the green nodes could be a robot, a sensor or some other IoT device. Each of these is connected to some network node and there is no pattern

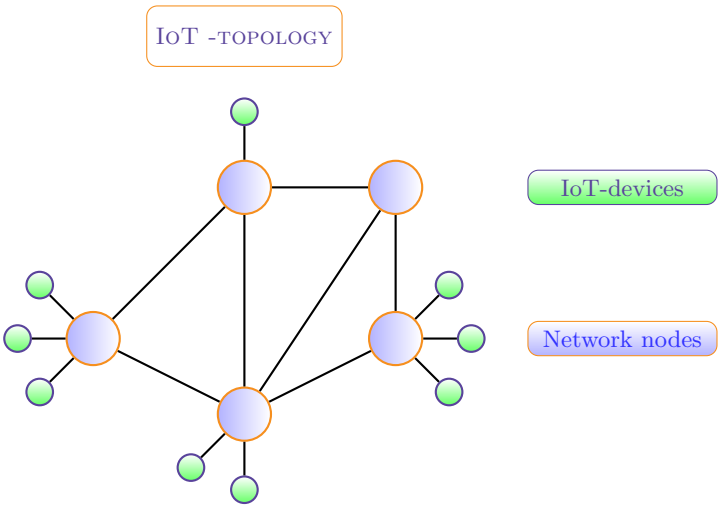

Figure 3.5 More advanced network topology

in how many devices are connected at each node. Furthermore, the IoT devices could be mobile and thus move around in the topology. What's more, the network nodes could also be heterogenous. In a 5G setting, some nodes may be eNodeB equivalents and others may be nodes deep in the core network. Taking this into account complicates the basic setup from Figure 3.1, but Calvin is built to handle it. Hence, even though the basic setup is very limited, most of the features can be immediately scaled using Calvin.

3.6 ROS

Whereas Calvin is not designed specifically for robotics there are several robotics frameworks, as pointed out by the ROS community [ROS, 2016b]. Although it was decided to use Calvin as the primary framework in this thesis a brief introduction to one of the robotics frameworks, namely ROS, will be given here.

ROS is an open source project for robotics that has gained traction in recent years [Michieletto et al., 2013] [Cousins, 2011]. Robotics is in many ways a multidisciplinary topic and ROS embraces that developing robot software is hard and tries to bring people from many disciplines together under one framework [ROS, 2016a]. Despite that ROS is short for Robot Operating System it is not scheduling and managing processes like a conventional operating system but rather works like a communication layer on top of other operating systems, thereby connecting robots, IoT devices and the cloud [A. Barbosa et al., 2015]. Although the core of ROS is the

communication layer, the main benefit is that the community continuously expands the set of libraries thereby providing the infrastructure to build robotics applications.

In terms of ROS in an industrial setting one drawback is that it has its roots in the academic community and is geared towards service robots rather than industrial robots. In order to address this issue an expansion, ROS-Industrial, has been developed [ROS-Industrial, 2016]. ROS-Industrial extends the ROS framework by providing libraries specific for industrial robots, such as interfaces to common industrial robots and enhanced path planning. Furthermore, using ROS-industrial allows for a combination of the high-level infrastructure provided by ROS while simultaneously drawing on the security of robot controllers [ROS-Industrial, 2016].

In comparison with Calvin, ROS/ROS-Industrial comes across as a more developed framework with many of the desired features for robotics. However, although not fully investigated due to the scope limits of this thesis, there might very well be limitations in the core of ROS that makes it difficult to fully utilize a dynamic behavior. Furthermore, since Calvin is a general purpose framework it is also possible that a future integration with ROS or some other robotics framework would be a worthwhile endeavor.

4

Implementation

In order to demonstrate how the proposed architecture could be implemented in practice a demo has been made. Since the architecture has been designed to be flexible and allow for modularization the aim of the demo is to show these features. Hence, the demo has been designed so that it is easy to see where additional features could be added and to somewhat mimic a possible 5G architecture with several computational nodes. In addition the demo involves a concrete task to be performed by the robot and calculations to be made in the cloud.

4.1 Hardware

The demo is run on the ABB Yumi robot connected to a local network where two additional computers also are connected. One of the computers is running Windows 8 with ABB RobotStudio providing an interface to change and modify robot programs. Thereby, making it is possible to work without using the somewhat limited flexpendant connected directly to the robot. The other computer runs an Ubuntu distribution and is used to run programs designed for Linux environments. The robot has no direct access to the internet but both the additional computers do.

4.2 Robot task

The robot task chosen for this demo is to solve a 2x2 Rubik's cube, see Figure 4.1. For those familiar with the 3x3 Rubik's cube this might seem like a considerably simpler problem but the 2x2 cube still has 3 674 160 possible permutations and maintains many of the tricky features of the 3x3 cube [*Jaap's Puzzle Page* 2016]. There are several solvers to be found online, however most of these are designed to be human friendly and use algorithms that require more rotations than necessary, but are perhaps easier to learn for a human. From a robotics point of view it is highly desirable to keep the number of rotations to a minimum and hence use a solver that produces such a solution. The minimum number of rotations has been proved to be

Figure 4.1 Rubik's cube, 2x2

at most 14 quarter turns [*Jaap's Puzzle Page* 2016]. Such solvers can also be found online and this demo uses a solver adapted from a competition [*Stack Exchange* 2016]. The solver is essentially a refined breath first search that produces a sequence of moves in a notation defined by the Rubik's cube community [Converse, 2016]. The reason why the breath first search algorithm has been refined is simply that a pure breath first approach generated too many combinations and was deemed too slow.

Although this task and its accompanying solver may seem very specific it is in fact very general. Considering the task from a broader perspective, it is essentially an optimization problem solved using brute force. It is not difficult to imagine that such problems occur regularly in a factory setting. For example packing a box in an efficient manner or optimizing the production schedule to maximize utilization and throughput would be examples of the so called knapsack problem which is NP-complete, meaning that there is no known algorithm that solves the problem correctly in polynomial time for all cases [Kellerer et al., 2004]. Using heuristic approaches to solving these kind of problems would be an ideal example of where pushing computations to the cloud is beneficial. Comparing the well studied and limited Rubik's cube problem, where the brute force approach needed some refinements, to a less well defined ad hoc optimization problem, that could occur in a factory setting, one realizes that computational power could provide significant performance improvements.

4.3 Controller implementation

As previously described it is possible to communicate with the robot on different levels and the second level, using socket communication, was deemed to be the most suitable for this thesis. In relation to solving Rubik's cube, this level fits very well since the problem is essentially a sequencing problem where a number of rotations should be done in the correct order. In addition, using the second level implies that a lot of the robot control and trajectory generation is handled by the on-board computer. Nonetheless, it is possible to make a case that the other two levels could be useful as well and the software architecture allows for an extension. From a controller point of view, the type of communication to some extent dictates what kind of choices have to be made and there are several decisions that still remain after deciding on communicating with the robot on the second level. The main concerns are what to send to the robot and where the information to send originates from. One can note that the task is deterministic given a specific starting position of the cube in the sense that a solution given by the solver will not change, although there will likely exist several solutions. Furthermore, the solver calculates the entire solution at once so the complete sequence could be sent to the robot immediately. However, this approach has some significant drawbacks if the task is considered from a more general point of view. The main drawbacks are an increased complexity of the robot program and less runtime influence from the controller resulting in a less adaptable architecture. On the other hand there are benefits of limiting the network traffic as well. Hence, the approach favored in this thesis and applied in the demo is to define some of the often used modules directly in the robot program. In terms of the Rubik's cube problem the sequence generated by the solver contains nine possible moves and these have been defined directly in the RAPID code in the robot program. This approach means that the solution provided by the solver is split up into individual moves and sent to the robot. Deciding on the level of modules to pre-define in the RAPID code should be done with some care and it is always possible to send each RAPID command individually. For the task at hand the reason for deciding to work on the "move"-level is that this level is the lowest level where commands will change from execution to execution.

4.4 Software implementation

The general architecture used in the demo follows from the considerations described in Chapter 3. However, this section describes the specific implementation in more detail. The backbone of the implementation is Calvin but due to practical considerations some additional software is also used.

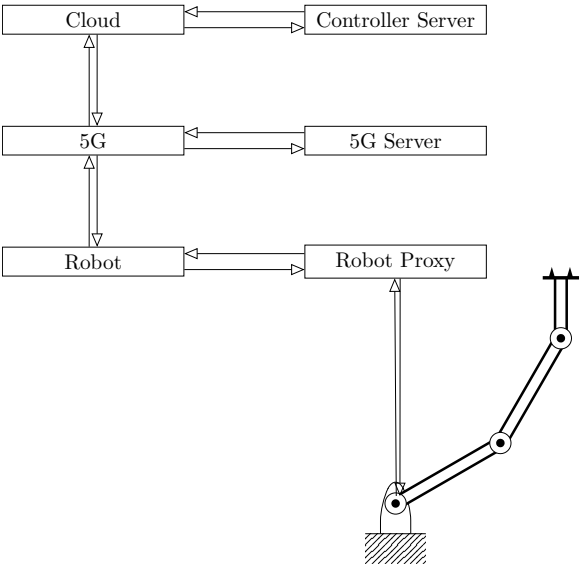

Figure 4.2 Schematic illustration of the application

Calvin runtimes

The first consideration for a Calvin implementation is to determine what a suitable configuration of Calvin runtimes would be. This is essentially a question about identifying where calculations should be made in various execution scenarios. One immediately concludes that one node will be needed close to the robot and one node will be required for the cloud calculations. Furthermore, since the demo is aimed at demonstrating not only how Calvin would work in a robotics scenario, but also to show how it fits with 5G, the setup of runtimes should be adapted for this purpose. Although 5G is not well defined at the time of writing there will most likely be one or several nodes in the 5G architecture where calculations could possibly be made. In order to demonstrate this, an additional node is added. Hence, three nodes have been identified as suitable Calvin runtimes and named Robot, 5G and Cloud respectively. A schematic representation of the setup is shown in Figure 4.2 where the Calvin nodes are the three nodes to the left. Each of the Calvin nodes communicates with a server outside of Calvin. The Cloud runtime is deployed at Ericsson's cloud servers and the other two runtimes are deployed locally at the Ubuntu machine in the local network connected to the robot. Ideally one would like to push the robotics runtime even closer to the robot and execute it directly on the operating system of the robot controller but this was deemed unfeasible for this thesis. The setup chosen is fairly limited but still captures the essential features of a distributed robotics architecture.

Calvin application

Similar to deciding on a suitable runtime configuration, the application running in Calvin has to be identified. In this particular case the functions of the core actors map perfectly to the Calvin runtimes which is mainly a result of that the application is kept to a minimum. First of all, the robot actor, is intended to be run on a runtime connected to a robot. Hence, this runtime has to be given a robotics capability. Secondly, the controller actor is normally run on the cloud server and this runtime should have a controller capability. However, the controller capability could in theory exist on other runtimes as well since it, unlike the robotics capability, does not require any hardware. For the purpose of the demo only the Cloud runtime is given the controller capability. Finally, the 5G runtime is given a 5G capability and a 5G actor is intended to run there. In this demo the 5G actor will serve as a dummy actor that could be replaced by another actor, e.g., a computer vision actor. Comparing this to Figure 4.2 one notices that the capabilities described above corresponds to the servers connected to the different runtimes.

Core actors When looking into how to construct the actors it was decided to keep the Calvin part to a minimum for this demo. By doing so, the application is very general in its nature and the components could be easily modified to fit other purposes. Furthermore, it was decided to build all the three core actors from the same template, thereby reducing the complexity. The template used is an already existing TCP client actor that comes with Calvin which means that the Calvin team could provide some additional support and fix a few bugs. The TCP client actor establishes a connection and forwards tokens to and from the server it is connected to. Essentially this is the behavior needed for the robot and 5G actors. Nonetheless, in order to map capabilities to requirements imposed by actors new actors had to be created. In the specific implementation the controller actor could have been extended to include more of the logic required to solve the task. However, in keeping with the general principle of building a generic demo it was decided that the same TCP client template should be used. The reasoning behind this decision is that it is desirable to be able to have different controller behavior on different runtimes but still let the runtimes have the same controller capability in Calvin, allowing Calvin to automatically manage the orchestration. Hence, by having a controller actor that communicates with the local host over a specified port, the behavior can be changed upon migration. Similarly the same effect could be achieved by migrating the robot actor to a runtime connected to another robot. As long as the second robot can perform the task, e.g., has that capability, the architecture is invariant to where an actor is executed. For a more elaborate account of the template actor and its accompanying socket client handler, see Appendix A.

Supporting actors In addition to the core actors there are a number of supporting actors with limited significance to the architecture. Most of them are used to generate tokens that provide connection information to the core actors. Furthermore,

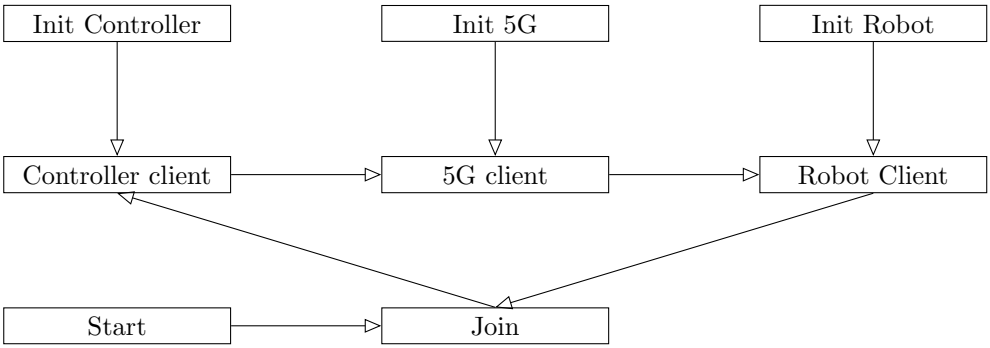

Figure 4.3 Full Calvin application

the application has been constructed so that a token will continuously be passed in a loop. In order to initialize the loop a start token is generated. Consequently, to achieve both the loop and start behavior an actor joining two streams is used. By designing the application as a loop, synchronization is achieved. This means that the robot will not be flooded with commands that it has no chance to execute. With the current setup, there will at most be one token waiting to be executed. The complete application is shown in Figure 4.3. As previously mentioned, synchronization is in general one of the major design issues with an asynchronous architecture. For example if a camera would be added to the Rubik's problem, one would have to make sure that the camera takes a picture at the right moment when the cube is shown to the camera while simultaneously keeping track of the orientation of the cube. The solution to this problem would be to pass additional synchronization tokens in different loops so that the actors are triggered at the appropriate time.

Communication outside Calvin

As described in the previous section the communication between Calvin and its surroundings is based on TCP. Furthermore, in order to achieve the desired behavior when the robot actor is migrated a proxy server is used at the robotics runtime. The proxy server's sole purpose is to ensure that the IP-adresses specified within Calvin can be constant regardless of runtime, e.g., "localhost". Similarily, the 5G runtime has access to an echo server. Finally, at the Cloud runtime, a simple server that passes the next robot move is implemented. All programs outside of Calvin, except the RAPID program, have been implemented in Python. In addition it is worth mentioning that the communication in Calvin is token based. However, in order to avoid problems when the various parts communicate with each other, all tokens leaving or entering Calvin through one of TCP clients are in the form of strings.

RAPID implementation

After investigating various ways of communicating with the robot and suitable protocols to use, the final implementation in RAPID is done with a very general program. The program is essentially a TCP server that listens for incoming connections, evaluates the string it receives and executes the command. Hence, any command that is supported can be executed remotely by just passing it as a string to the robot. For the purpose of the demo this approach works well but it has some significant drawbacks in terms of that it is possible to pass commands that are not valid, causing runtime errors. Furthermore, it opens up a rather large security hole since no security features at all are added. Hence, it would be a clear improvement to use a marshalling protocol with security features. Unfortunately, the marshalling protocols investigated, namely Labcomm, was not up to date and it was deemed that the added benefit of updating it was not yielding sufficient returns for this thesis to warrant the effort.

Miscellaneous

In order to facilitate a repeatable and portable development environment Docker is used. Docker is built around containers which is essentially a light weight version of a virtual machine [Docker, 2016]. Although this approach is useful there are a few considerations for the task at hand. Since the runtimes have to be able to communicate with the host running docker and the network connections, Docker has to be setup properly to forward ports. Some of the docker scripts included in Calvin help with this, however, they are not designed for opening additional ports and making sure docker handles this traffic correctly. In addition the firewalls and shell scripts differ on the Ubuntu machine and the workstation used to write the programs, which uses Fedora. This causes quite a lot of problems with the portability of the docker images.

Limitations of the demo

The demo is developed in order to be very flexible and possible to adapt. Although, this is one of its main strengths it does also mean that there is a lot of room for improvement when it comes to solving the specific task. There are several parts of the demo that could be easily improved, such as adding computer vision, enhanced trajectories for the robot, better security etc. In addition the demo could also be expanded to use several robots and different controllers connected to each of the runtimes. The architecture does however provide a framework for future work, where each of these parts could be gradually developed.

5

Results

After reviewing relevant software, studying literature and constructing a demo the following chapter addresses the problem formulations posed in Chapter 1.2. However, prior to addressing the three research questions in the problem statement it can be concluded that a working demo has been constructed. The demo has all the essential components for a distributed robotics architecture. Although, the demo has in parts been restricted in order to remain within a reasonable scope for the thesis work it captures many of the challenges inherent in constructing such an architecture. Furthermore, the construction of the demo has highlighted several practical problems with limited theoretical relevance. Hence, one of the results from this work is to acknowledge that the approach of combining Cloud Technology, robotics and telecommunications is complicated. Thereby also acknowledging that good frameworks has the potential to improve productivity in the field. Nonetheless, it should be stressed that the architecture implemented in the demo has been a success and could form the basis for future research.

In Section 1.2 the following research questions were stated and will be addressed in turn below.

1. What is a suitable software architecture for cloud robotics using 5G?
2. What is a suitable controller architecture for cloud robotics using 5G?
3. What requirements does cloud robotics impose on 5G?

5.1 Software architecture

During the work process, the following results have been obtained in terms of how a suitable architecture for cloud robotics using 5G could be designed.

Basic Actor model

The proposed software architecture is based on the actor model which from the work done in this thesis seems to be a promising architecture. Throughout the work it has

been noted that there is a constantly evolving need for new components to be integrated. The Actor model provides possibilities for simple additions since the actors work in isolation. Furthermore, it has been recognized that it often is desirable to be able to use software originally not intended to be used in an actor. It should therefore be easy to convert existing software into actors thereby augmenting the usefulness of the framework. Hence, actors that provide a universal interface to other software are promoted. Moreover, many advanced robotics applications often require input from various sensors and it would be very beneficial if the infrastructure for this was present. Therefore, it is suggested that standardized libraries should be made available and used whenever possible. Once again, the Actor model fits well due to the isolated character of the actors which allows for efficient code reuse.

Augmented Actor model

Although, the basic Actor model provides many useful features the most interesting aspects come from the dynamic behavior it enables. Being able to map actor requirements to capabilities of the environment in where it should run provides a good foundation. However, for cloud robotics the need for more advanced dynamic behavior becomes apparent.

Capabilities The notion of a capability needs further refinement. Assigning a capability to a robot based purely on its ability to perform a certain task is insufficient as the quality of the performed task may vary in both execution time and proneness to errors. One solution to this problem would be to construct various capabilities based on the quality. This approach would however be counter productive as it would remove any possibility of an autonomous dynamic behavior. Hence, one of the results in this thesis is that for cloud robotics applications it should be possible to rank the capabilities. With ranked capabilities it would be possible to dynamically pick the most appropriate configuration during runtime. In the implementation phase this issue was partly solved by assigning actor requirements in Calvin that mapped to capabilities that in turn used a TCP connection to servers running on the local host. As an actor is migrated the reference to "localhost" will be different on different runtimes. By doing so it is possible to alter the behavior of the actor depending on which runtime it was running on without altering the application structure at runtime. This does however not solve the issue of grading the different capabilities.

Dynamic behaviour The ranking of the different capabilities might very well change during runtime. Some of the reasons why the grading might change is not specific to robotics, for example variations in load on the different runtimes causing variations in execution time. Other aspects are robotics specific. One such aspect is that robots, as opposed to a conventional program, always have a state which has to be accounted for. As a practical example of this, consider that a robot will require a tool change before performing a certain task whereas another identical robot already is equipped with the relevant tool. The robot requiring the tool change still

has the capability but it will certainly be slower at executing the task. In addition to the issues pointed out above there are a number of circumstances which are more business oriented, such as operating costs and wear on the robots. This reasoning points to that the grading of capabilities should be done dynamically.

Penalizing poor performance A distributed architecture, which using the cloud entails, will add latency as compared to running local applications. This is further amplified by using a telecommunications technology that implies using a wireless transportation medium. The added latency will vary depending on which nodes one refers to and moreover 5G will with great certainty include some sort of random access scheme akin to that of 4G causing variations in latency. These issues has to be handled dynamically by the software architecture in order to maintain the benefits of having a distributed architecture. For example, it is of no use to have fast cloud based calculations if the latency in the infrastructure consumes all of the added performance. The solution suggested in this thesis is once again a dynamic handling of the problem, where the remote capabilities are penalized for the added latency. Exactly how the penalty should be constructed will be dependent on the time sensitivity of the application and its controller. In the case where the application is designed to be insensitive, by performing sequential operations, the main consideration will instead be to avoid stalling by keeping the latency below the time required for one execution step in the sequence.

Lost connection The important case when the connection is completely lost has to be accounted for. This case provides a significant challenge for the Actor model since the the actors have very limited knowledge about the other actors. For example, consider an application where a controller actor runs on a cloud server and the connection is lost. In such a case there is no clear way for the application to proceed since it receives no new instructions and the actor can not be migrated either. Considering that robots could be potentially harmful for their surroundings, if left unmonitored while executing, it is evident that some sort of fallback mechanism is essential. Whereas the specific behavior of this mechanism will vary depending on the application the architecture should handle the problem in an appropriate way. In regards to this one can note that compared to many other distributed applications robotics applications has the feature that one node is significantly more important than the others, namely the node closest to the robot. Without this node the entire robot application fails and hence it should preferably be placed as close to the robot as possible in order to minimize the risk of a connection failure between the robot and the node. If possible it is recommended to place it on the on-board computer of the robot. The interesting part of this node is that it is not only the most critical node but it is also a node where it is desirable to do few calculations. This insight imposes some additional constraints on how to handle the situation when a connection is lost. One possible solution would be to simply mirror all actors on the robot node and thereby be able to continue executing. This approach has a three-fold problem. One of the problems is the capacity constraint of the node which is a deterrent to super-

fluous execution . The other problem is that the actors would not necessarily be in sync and could therefore send ambiguous messages. The third problem is that the application would not recover in the sense that it might very well be possible to run the actors on some other node instead. The solution proposed here is instead to use a more refined technique that draws on the dynamic behavior. Instead of mirroring the entire application only the minimum required information, such as the state of the robot, is kept at the robot node. In case of a lost connection, the lost actors are redeployed and a new instance of the application created from the remaining actors and the redeployed actors in a sort of bootstrapping process. The application will then be dynamically dispersed as the runtime environment sees fit while old actors that might get reconnected will not interfere. Taking this approach ensures that the application will run at the best possible performance regardless of which connections fail.

Synchronization

With an Actor model, synchronization becomes an issue. Since actors are built to work in isolation there is no natural synchronization built into the model. The proposed architecture therefore implicitly suggests that these issues are to be handled outside of the model, e.g., when building applications. The developer will simply have to ensure that the application is built to synchronize itself at the appropriate moments. This can be achieved through building applications using actors that require input from several other actors before triggering an action. This is a significant drawback of the proposed architecture and it is important that deadlocks are avoided. However, it is the author's opinion that it is fairly easy to avoid these considering that the applications can be constructed at a relatively high level. Although, taking this approach will very likely have a performance deterring effect on real-time constrained applications.

5.2 Controller architecture

As previously discussed, there is one major decision to be taken regarding the architecture of the controller, namely at which level to control the robot. The work done in this thesis points towards that it is strongly recommended to use as much of the on-board features as possible in the vast majority of cases. The reasoning behind this is that several control loops are already implemented by the robot manufacture and there simply is no reason to interfere with these in most cases. Hence, the proposed architecture relies heavily on using many of the existing controllers for the low level tasks. Instead the proposed architecture focuses on sequencing as the main level of control. The reasoning behind this recommendation is that sequencing is far less sensitive to the asynchronous behavior of the software architecture. Besides, even in a sequencing based application it is possible to change the granularity of the steps in the sequence. Reducing the step size would increase the influence

from a remote controller but could interfere negatively with the trajectory planning of the on-board computer. Furthermore, there are plenty of applications based on sequencing that would benefit from a distributed architecture. Consequently, moving forward it is easier to first solve many of the challenges connected to cloud robotics and 5G in a less sensitive environment. Nonetheless, in the cases where it is actually desirable to make alterations to the control signals it is still fully possible. However, even in these cases a sequencing approach should constitute the backbone of the application and alterations should only be made in the critical areas. In regards to this it should also be mentioned that even a small robot movement takes relatively long time compared to transmitting data, which is in the order of milliseconds. Hence only the applications with real-time constraints are affected.

In terms of the relation to the proposed software architecture the controllers are mainly affected negatively in the situations where the connection is lost. In particular controllers relying on old information to calculate the next control signal. The reason for this is that the proposed software architecture implies a bootstrapping approach where old information is lost if a connection fails. On the other hand, the software architecture also has some positive benefits for controllers running under normal circumstances. Especially the increased access to storage, computational capacity, and common information in the cloud has the potential to improve controllers significantly. Theoretically, a controller could draw on the data accumulated over a long period and even on data from similar processes.

As a final remark on the controller architecture it should be mentioned that it is important that the robot node has some sort of controller capability in order to ensure operation in a situation where it is the only accessible node. The motivation for this is, as previously mentioned, that robots should not be left entirely unmonitored.

5.3 Requirements on 5G

In terms of requirements imposed on 5G by cloud robotics, there are a few remarks to be made. First of all, the proposed software and controller architectures in this thesis suggest that many interesting applications can be built based on sequencing. Taking this approach means that some of the toughest latency demands are avoided. As previously mentioned, the reasoning behind this approach is that there seems to be a lot of interesting applications that can be built and investigated further before pushing the latency demands to a critical level. Furthermore, it is very difficult to come up with situations whith real-time constraints that require wireless communication and even harder to find situations where the preferred wireless solution uses licensed spectrum and random access schemes. Nonetheless, if one finds such applications it is worth knowing that the frequency used for altering command signals to the motors is 250 Hz, implying a maximum total delay of 4 ms for calculating the control signal and sending it to the robot. It should be mentioned that depending on the specific application it might be possible to do some dead-time compensation

to make it less sensitive. However, considering that a simple ping command from the LTH network to the Ericsson servers takes 22 ms using a wired communication channel it seems unlikely that a cloud solution for these applications can be recommended. On the other hand, it is possible that some node in a 5G architecture could be used. For example the radio node could be of interest as the least common denominator if several devices operate within the same cell. For example, this node could be used for calculating common trajectories or image recognition. Hence, a result found in this thesis work is that the 5G architecture should allow for using third party software if it is to be compatible with real-time constrained robotics applications. For the sequencing based approach favored in this thesis, other aspects are in focus. As discussed above, some of the major problems with the software architecture proposed is that it is sensitive to lost connections and to some extent also for variations in latency. Therefore, stability is of the uttermost concern for robotics in 5G. Being able to, with a very high probability, guarantee that the latency is below a certain threshold would make it considerably much easier to detect when it is appropriate to go into fallback mode. With a high level of variations this threshold would either have to be set too high, impeding performance, or cause undesirable fallback behavior in cases when it is not needed. Considering that a pure random access scheme could cause problems during peak loads another result is that 5G should have features for prioritizing traffic. This prioritization scheme should at the very least allow actors to be migrated towards the robot with a very high priority. Finally, the security aspects connected to a wireless technology deserves special contemplation. Once again, remember that large industrial robots are inherently dangerous for humans to be around and could also cause severe material damage. In addition to this, if robots are used for manufacturing there is a risk that the products could be altered to behave in an dangerous fashion, e.g., consider a car breaking down at high speed due to a faulty component. Although security aspects are well known problems it should be stressed that if 5G is to be used for industrial robots, communications security should be treated with the outmost respect.

6

Discussion

After investigating the architecture and requirements for cloud robotics intended for 5G as well as implementing a demo the experiences gained point towards some interesting topics for discussion. First of all, one should note that the general approach of combining the three areas is fully feasible and an interesting area for research. However, as pointed out by the developers of ROS, the problem domain of robotics is complicated and adding the cloud and telecommunications does not make it any less complicated. At the same time the benefits that can be envisioned are extremely appealing and an appropriate way of limiting the complications should be valuable. One such possibility has been investigated in this thesis, namely using Calvin, but there are a multitude of different ways of achieving more or less the same results. In other words, building a cloud robotics architecture for 5G is in itself not revolutionary. This observation prompts another insight, specifically that the difficultly of finding an architecture is not necessarily a technical one, but rather a business case. The argument for this preposition is that the complicated nature of the task results in a poor scaling. Furthermore, as previously pointed out the adaption rate of smart manufacturing is dampened by difficulties in deeming the investments as profitable. Hence, finding a solution that is both technically sound while allowing a working business model is the real problem at hand. The approach favored in this thesis could provide some interesting opportunities, particularly considering that Calvin is being built to be dynamic. To further discuss these opportunities, a number of scenarios will be described. Though, prior to dwelling into these scenarios a few remarks about manufacturing industries in a generic sense will be made.

6.1 Manufacturing industries

The following assumptions about a generic manufacturing industry are made.

1. Historic strength is in the production and logistics connected to manufacturing, not in IT.
2. Significant investments have already been made in machinery.

3. Specialized in producing a limited set of products.
4. Heterogeneous interfaces with machinery and other systems.
5. Costly if production is stopped.

One could certainly argue with the assumptions but the intent is only to provide a foundation for the discussion below. If the assumptions are invalid for the particular industry at hand the arguments presented may not always be valid. Nonetheless, having a foundation to build on adds clarity and has therefore been included.

6.2 Scenarios

In this section scenarios illustrating how the proposed architecture can be used and what benefits it brings are presented. Rather than being purely technical they intend to demonstrate usefulness in a setting that is business friendly albeit technical. Furthermore, they attempt to stretch beyond current limitations.

Performance

The main driving forces behind using cloud technology are to utilize the increased computational power and economic efficiency. In the case of robotics the situation is however not always so clear. First of all, some applications may be very sensitive to delay so that the extra latency makes it useless or even harmful to do calculations in the cloud. However, even in the less sensitive cases it is not always clear that a cloud solution is preferable. As postulated above, it is very costly if production is stopped and taking the risk of being dependent on a remote connection is not too appealing. Nonetheless, one would like to find a way to utilize the benefits of the cloud. The proposed architecture enables an increased performance while limiting the negative side effects. There are in particular two cases that should be mentioned. The first case is when the default setup is to use the cloud. If the connection fails or the latency becomes too high, the actors are migrated (or recreated) at a node closer to the robot. By taking this approach, the risk associated with the connection is limited while the cloud is being utilized. The second case is the opposite, where the default configuration is to run the calculations locally and migrate to the cloud if the load on the local node is too high. In taking this approach the sensitivity to limited computational power at the local nodes is handled. However, from a point of view of the proposed architecture, both these cases are the same. The only difference is how the dynamic assessment of capabilities is done.

Stability

Using the cloud inherently adds increased risk for instability due to the magnified dependency on an internet connection. However, there are other factors that could offset this problem and improve the total stability of the plant, thereby reducing

costly downtime. First and foremost pushing calculations to the cloud would reduce the complexity within the factory and make it easier to replace parts and upgrade software. One could even consider a situation where a component in the factory breaks down and the remaining working machinery automatically organizes themselves to build a replacement part. Furthermore, one can easily imagine a scenario where the monitoring of the factory is done remotely. In addition, the vastly improved computational resources makes it feasible to make a lot more simulations, e.g., simulate the wear on the machinery. Similarly to the performance argument, the dynamic behavior of the proposed architecture could also improve the stability of the plant by penalizing instability in the connection. The addition of 5G friendliness already at the architectural level expands the possibilities for a seamless transition to a backup strategy in case the fixed wired connection fails.

Mobility

As has been pointed out earlier in this thesis, finding a clear cut example of when 5G truly adds value in a factory setting is somewhat difficult. The benefit of using a wireless channel would of course be to avoid the cables but there already exist other wireless solutions that could very well be used to create wireless communication in a factory. However, the case where one would like to move the factory regularly is a case where the existing solutions may be insufficient. Consider a robotics cell used for construction on site. It could for example help to lift heavy parts or construct custom components that are hard to transport. In this scenario having a communications infrastructure that can be accessed immediately regardless of location without any need to build additional infrastructure adds value. At the same time, mobility induces additional uncertainty as the communications conditions may vary from location to location. Therefore, the need to have a flexible architecture is further enhanced. What's more, mobility intrinsically means that robots have to keep track of each other and their surroundings. This in turn suggests some sort of coordination and an opportunity to draw on information gathered by other sources, perhaps by using a map built by another robot. Hence, joint storage and computations would come in handy and the proposed architecture, aiming to use nodes in a future 5G infrastructure is advantageous.

Streamlining

Taking an approach of studying the competitive advantages, perhaps by using Porters five forces framework, in order to improve firm strategy it is possible to identify some further beneficial aspects of the proposed architecture. Consider a generic manufacturing firm. As assumed in Section 6.1 such a firm has its strength in the production and the logistics connected to this part of the workflow. Their strengths are however typically not in IT, which could be characterized as a supporting function. In keeping with the strategic concept of focusing on the areas where one has an advantage the strategic recommendation would be to outsource IT. How-

ever, with an increasing desire to integrate IT with production in accordance with the smart manufacturing paradigm these resources become more tightly coupled. With the proposed architecture a decoupling of IT and the conventional production is possible, thereby allowing for a more streamlined corporate strategy. Although there might be other reasons why one would not like to outsource, the proposed architecture does at least provide a possibility to do so.

Integration

One of the main ideas behind smart manufacturing is to utilize the advances in information technology in order to improve production capabilities. Essentially this means drawing on the increased capacity to accumulate data and to draw conclusions from this data to boost performance. Whereas, some of the data is likely to be gathered from the existing machinery, other parts might come from external sources, for example IoT devices, Enterprise Resource Planning systems (ERP) or Customer Relationship Management systems (CRM). The architecture proposed in this thesis prepares for an environment with an increasing access to new data by employing a flexible operations scheme based on the actor model. Since the actors are isolated new ones can be created without altering the old ones and the token based system provides a clean interface allowing a gradual development. Consider for example an actor that uses data from the ERP system to provide a robot cell with instructions about what to make, and other actors handling the actual execution. In such a scenario all the actors could be developed independently. Yet another example would be the countless IoT devices projected to appear in the future. Developing an actor for these would immediately create a clear cut interface allowing for easy integration. In addition, the existing non-IoT devices should also be possible to integrate by letting their signals pass through an appropriate actor. Seeing that manufacturing industries tend to have plenty of old equipment which is expensive to replace this should be a welcomed possibility. Furthermore, the proposed architecture would allow for a seamless addition of new components that perform the same function as old ones. One might even consider leaving old components and only assign a higher capability grading to the new ones, thereby adding redundancy and fallback options without additional cost.

Efficient allocation and resource usage

By providing a dynamic infrastructure the efficiency in terms of resource allocation could be improved. In a setting limited to a factory this would apply both in terms of being able to make better use of data from the ERP system to optimize the production schedule as well as handling unexpected changes due to machines breaking down. Consider for example that a new order is entered into the ERP system. In such a case it would be possible, and maybe even likely, that the optimal sequence to handle production changes. From a mathematical perspective adding an additional order is equivalent to imposing a new restriction in an optimization

problem. Solving such problems frequently could require substantial computational effort where endorsing the cloud would be useful. Furthermore, one could also easily imagine a scenario where surplus capacity in a factory is sold in order to constantly utilize the machinery to its full extent. In a more market oriented approach entire factories could be built to produce a variety of products on demand. Taking this thought further one could imagine that the production mode as we know it today would be entirely replaced by multipurpose factories, producing virtually all industrial products for the local market. Hence, the dynamic infrastructure suggested has the potential to vastly change the conditions for industry. Admittedly this might at first glance seem to be far into the future but mass customization is as previously mentioned a growing trend where this might be a competitive solution.

Improved markets

The transaction cost theory suggests that the major impediment to purchasing products and service in the open market, is that such a transaction is associated with transaction costs. In the case of manufacturing one can envision the costs connected to finding a suitable manufacturer, establishing specifications, negotiating contracts, etc., would be significant. For this reason purchasers might be reluctant to change suppliers or avoid using the market for smaller series. Since the proposed architecture provides a clean interface transaction costs would be lowered thereby improving the function of the market. For example, if one has an application, a set of actors and connections, that assemblies a particular product the dynamic mapping of capabilities and actors would immediately answer the question of who can produce a certain product. Taking this yet one step further, the application could be created from some other standardized format, such as a CAD-drawing, before its actors would be mapped to capabilities. Moreover, information leakage could be decreased since it would be possible to enclose some of the sensitive information, e.g., control parameters, in actors that are never run remotely. Furthermore, apart from the advantages that can be achieved from a manufacturing firm's point of view, there is also a business opportunity in providing the infrastructure to facilitate the improved markets. Similarly to app-stores for smartphones one can easily envision a market place for these kind of services where a third party would guarantee that both purchasers and suppliers follow through on their commitments.

Learning and reuse

Although manufacturing industries are used to drawing on the advantages of economics of scale in their production, those aspects are mainly on the product level and not on the production equipment level. Even if a factory is super efficient and produces large quantities, it typically does so with a limited set of machinery. Hence, by connecting machinery to the cloud it is possible to draw on the experience gathered from similar machinery in other non-competing industries. For example, a welding machine might be used in different industries without the final product being

even remote substitutes. Hence, the architecture forms the foundation of economics of scale at a higher level than before. Furthermore, connecting more IoT devices with the cloud enables an augmented capability for gathering data. Combining the access to the cloud with vast quantities of new data gives excellent opportunities to use emerging technologies, such as machine learning to find improved solutions. Once again invoking the argument of competitive advantages some industries would be inclined to use these possibilities. A solution using the combined data of several industries could be leveraged by a firm either as a means of neutralizing competitor advantages or to build an advantage of their own by having a capability to cooperate with others in non-critical areas. In addition to benefiting from a combined pool of data, there is also a clear profit to be made from reusing components. Although an application is likely to be specific, the actor model allows for an efficient reuse of components. For example applications using image processing could use the same actor, thereby spreading the costs of development and maintenance across several applications. This fits very well with the previous argument of increasing the effectiveness of markets. As opposed to conventional software development it would be easier to keep parts hidden by only providing the capability on controlled runtimes. What's more, during the work with the thesis at hand, it has become clear that although a framework like Calvin aims to make it easy for developers to create applications, there still is a considerable effort needed to get over the hurdle of getting acquainted with the framework. Nonetheless, there is a definite benefit once one is over that hurdle. This insight combined with the ability to reuse existing components suggests that there could be a very lucrative market for creating applications from existing components. Preferably, this should of course be automated as well, but even using application engineers could be a profitable endeavor.

7

Conclusion

In this thesis an architecture for cloud robotics for 5G has been investigated. There are many interesting research questions that can be found within this area, but this thesis attempts to provide some answers to the following questions.

1. What is a suitable software architecture for cloud robotics using 5G?
2. What is a suitable controller architecture for cloud robotics using 5G?
3. What requirements does cloud robotics impose on 5G?

In addition to addressing these issues a demo has been implemented as to work as a proof of concept. The demo successfully features the core functionality of the proposed architecture and hence forms a good foundation for future research. Furthermore, from the experience gained while implementing the demo it was concluded that the implementation could be done in multiple ways. Nonetheless, the work done proves that the Calvin framework is one of those possible solutions. However, it was also concluded that the main problem is to find an architecture that is business friendly while still benefiting from all the desired features in cloud robotics. Hence, this thesis also shows a number of use cases where the proposed architecture could be utilized to leverage its strengths.

7.1 Software architecture

The software architecture proposed in this thesis is an augmented Actor model. By using the standalone characteristics of actors and the dynamics of capability mapping it has been shown that such a model fits well with cloud robotics. However, it has also been indicated that there are quite a lot of desirable features which require future work. Even so, the proposed architecture takes into account these possibilities and is built to easily be extended. The implementation in Calvin shows that it is already feasible to construct robotics applications based on an Actor model and it has also been noted that Calvin is built from the start to be dynamic, thereby having

a foundation that encompasses the desired features for cloud robotics. In conclusion, the proposed architecture shows great promise but future work is required. Some of this work is a fairly straightforward implementation whereas other areas will require significant research.

7.2 Controller architecture

When designing a suitable controller architecture it was noted at an early stage that some important decisions would dictate how the work would proceed. Particularly the decision on which level to communicate with the robot was of significance. However, in considering possible demo applications it was concluded that many interesting robotics problems could be handled at a sequencing level and the attention in this thesis has therefore been focused on addressing issues at this level. Considering that there are plenty of interesting problems left to solve, this approach is deemed to be successful. The effect though, is that many of the control loops will be handled by the on-board computer of the robot. From a controller design point of view it has therefore been important to maintain the possibility to extend the architecture. Nonetheless, some results regarding the controller architecture for sequencing based cloud robotics have been found. First of all, the cloud architecture allows for controllers that draw on a lot more data than conventionally. Secondly, the software architecture and network topology make it possible that the controller loses contact during execution, which has to be handled. Summing up, one could say that the controller architecture has to be adapted to fit the software architecture and to make sure to use the cloud capabilities in order for the combined architecture to add value.

7.3 Requirements on 5G

Since mainly the sequencing aspects have been considered, the really tough demands on 5G have not been fully investigated. It should be mentioned though, that the signals can be sent to the robot at approximately 250 Hz in case a more real-time application is considered. Despite that mainly sequencing has been the focus it has been concluded that stability is of the outmost concern and that being able to do computations on various nodes in the 5G architecture is highly desirable. Furthermore, the experiences show that it is quite difficult to pinpoint exactly in what kind of situations 5G would be an immediate difference maker in terms of cloud robotics since many applications could already be built with existing technology. Hence, in conclusion, at the time of writing, 5G seems to be a "nice to have" feature in cloud robotics and many challenges with a greater performance impact can be found elsewhere.

7.4 Concluding remarks

As described above, the work done in this thesis investigates how an architecture for cloud robotics using 5G can be constructed. It has been shown how such an architecture can be constructed, both on a conceptual level as well as a practical level by implementing a demo. The suggested architecture and the framework used, namely Calvin, show great promise for future work.

8

Future research

Throughout the thesis several possible improvements have been pointed out. Some of these improvements are straight forward implementation issues which should be fairly easy to deal with given the appropriate time. There have also been some issues which might be trickier but still have to be considered to be of limited theoretical value. However, there are some areas worth mentioning as possible areas for future research. First of all, exactly how the dynamic mapping of capabilities should be done needs to be investigated further. This work might consider suitable penalty functions and procedures for updating the application topology. Secondly, it would be of great interest to consider the production possibilities further. Similarly to the Production Possibility Frontier (PPF) found in economic literature, a set-theory based approach for determining what can be produced given a set of capabilities could be developed. Thirdly, in order to fully utilize 5G it would be beneficial to thoroughly study the real-time aspects and how these can be addressed.

Bibliography

A. Barbosa, J. P. de, F. P. C. Lima, L. S. Coutniho, J. P. R. R. Leite, J. B. Machado, C. Valeria, and G. S. Bastos (2015). "ROS, android and cloud robotics: how to make a powerful low cost robot". In: *Advanced Robotics (ICAR), 2015 International Conference on*, pp. 158–163. DOI: 10.1109/ICAR.2015.7251449.

ABB (2010). *Technical reference manual RAPID Instructions, Functions and Data types*. URL: https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf (visited on 09/28/2016).

ABB (2015a). *ABB introduces YuMi®, world's first truly collaborative dual-arm robot*. URL: http://www.abb.com/cawp/seitp202/01ce5df31a94676bc1257e260011d5b0.aspx (visited on 09/28/2016).

ABB (2015b). *Robot Web Services*. URL: http://developercenter.robotstudio.com/Index.aspx?DevCenter=Robot_Web_Services (visited on 10/01/2016).

ABB (2015c). *Utökade möjligheter med ABB:s program för robotstyrning*. URL: http://www.abb.se/cawp/seitp202/21cf9a10157d4c0fc1257dda003874dd.aspx (visited on 10/01/2016).

Abed, G. A., M. Ismail, and K. Jumari (2012). "The evolution to 4G cellular systems: architecture and key features of lte-advanced networks". *IRACST – International Journal of Computer Networks and Wireless Communications (IJCNWC)* **2**:1, pp. 2250–3501.

Agha, G. (1986). *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA. ISBN: 0-262-01092-5.

Andrews, J. G., S. Buzzi, W. Choi, S. V. Hanley, A. Lozano, A. C. Soong, and J. C. Zhang (2014). "What Will 5G Be?" *IEEE Journal on Selected Areas in Communications* **32**:6, pp. 1065–1082.

Angeles, S. (2013). *Cloud vs. Data Center: What's the difference?* URL: http://www.businessnewsdaily.com/4982-cloud-vs-data-center.html (visited on 09/30/2016).

Anthony, L. (2012). *Choosing between License and Unlicensed Spectrum*. URL: https://lesleyanthony.wordpress.com/2012/05/12/choosing-between-license-and-unlicensed-spectrum/ (visited on 10/01/2016).

Bazelon, C. (2009). "Licensed or unlicensed: the economic considerations in incremental spectrum allocations". *IEEE Communications Magazine* **47**:3, pp. 110–116. ISSN: 0163-6804. DOI: 10.1109/MCOM.2009.4804395.

Besanko, D., D. Dranove, and M. Shanley (2007). *Economics of strategy*. Businesses in the book. John Wiley & Sons. ISBN: 9780471679455.

Brandl, D. (2016). *The future control system environment, as envisioned by the exxonmobil next generation control system pilot project*. LCCC Workshop.

Calvin Repository (2016). URL: https://github.com/EricssonResearch/calvin-base (visited on 09/28/2016).

Cerwall, P., M. Byléhn, S. Carson, R. Möller, S. Bävertoft, J. S. Sethi, and P. Jonsson (2016). "Ericsson Mobility Report - ON THE PULSE OF THE NETWORKED SOCIETY". URL: http://www.ericsson.com/res/docs/2016/mobility-report/ericsson-mobility-report-feb-2016-interim.pdf (visited on 09/28/2016).

Charousset, D., T. C. Schmidt, R. Hiesgen, and M. Wählisch (2013). "Native actors: a scalable software platform for distributed, heterogeneous environments". In: *Proceedings of the 2013 Workshop on Programming Based on Actors, Agents, and Decentralized Control*. AGERE! 2013. ACM, Indianapolis, Indiana, USA, pp. 87–96. ISBN: 978-1-4503-2602-5. DOI: 10.1145/2541329.2541336. URL: http://doi.acm.org/10.1145/2541329.2541336.

Chiosi, M., D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, D. C. Cui, D. H. Deng, J. Benitez, U. Michel, H. Damker, K. Ogaki, T. Matsuzaki, M. Fukui, K. Shimano, D. Delisle, Q. Loudier, C. Kolias, I. Guardini, E. Demaria, R. Minerva, A. Manzalini, D. López, F. J. R. Salguero, F. Ruhl, and P. Sen (2012). *Network Functions Virtualisation*. URL: https://portal.etsi.org/NFV/NFV_White_Paper.pdf (visited on 10/01/2016).

Coase, R. H. (1937). "The nature of the firm". *Economica* **4**:16, pp. 386–405. ISSN: 1468-0335. DOI: 10.1111/j.1468-0335.1937.tb00002.x.

Converse, J. M. (2016). *J. Converse*. URL: http://w.astro.berkeley.edu/~converse/rubiks.php?id1=basics\&id2=notation (visited on 09/28/2016).

Cook, J. S. (2015). *Origin of the word robot*. URL: http://www.roboticstrends.com/article/origin_of_the_word_robot (visited on 09/30/2016).

Corke, P. (2011). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg. ISBN: 9783642201431.

Cousins, S (2011). "Exponential growth of ROS [ROS topics]". *IEEE Robotics & Automation Magazine* **18**:1, pp. 19 –20. DOI: 10.1109/MRA.2010.940147.

Davies, R. (2015). *Industry 4.0 Digitalisation for productivity and growth*. URL: http://www.europarl.europa.eu/RegData/etudes/BRIE/2015/568337/EPRS_BRI(2015)568337_EN.pdf (visited on 09/29/2016).

Delforge, P. and J. Whitney (2014). *Data center efficiency assessment*. Issue Paper.

Department of Automatic Control LTH (2016). *Robot arm controller*. Intranet.

Docker (2016). *Docker*. URL: https://www.docker.com/what-docker\#/VM (visited on 09/28/2016).

Eker, J. and J. W. Janneck (2003). *CAL Language Report Specification of the CAL Actor Language*. Tech. rep. UCB/ERL M03/48. EECS Department, University of California, Berkeley. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2003/4186.html.

GSA (2015). *The Road to 5G: Drivers, Applications, Requirements and Technical Development*. URL: http://www.huawei.com/minisite/5g/img/GSA_the_Road_to_5G.pdf (visited on 09/30/2016).

Hasan, M., E. Hossain, and D. Niyato (2013). "Random access for machine-to-machine communication in lte-advanced networks: issues and approaches". *IEEE Communications Magazine* **51**:6, pp. 86–93. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6525600.

Hewitt, C. (2007). "What is commitment? physical, organizational, and social (revised)". In: Noriega, P. et al. (Eds.). *Coordination, Organizations, Institutions, and Norms in Agent Systems II: AAMAS 2006 and ECAI 2006 International Workshops, COIN 2006 Hakodate, Japan, May 9, 2006 Riva del Garda, Italy, August 28, 2006. Revised Selected Papers*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 293–307. ISBN: 978-3-540-74459-7. DOI: 10.1007/978-3-540-74459-7_19.

Hewitt, C. and H. G. Baker (2002). "Laws for communicating parallel processes." In: *IFIP Congress*, pp. 987–992. URL: http://dblp.uni-trier.de/db/conf/ifip/ifip1977.html#HewittB77.

Hewitt, C., P. Bishop, and R. Steiger (1973). "A universal modular actor formalism for artificial intelligence". In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*. IJCAI'73. Morgan Kaufmann Publishers Inc., Stanford, USA, pp. 235–245. URL: http://dl.acm.org/citation.cfm?id=1624775.1624804.

IFR (2012). *The continuing success story of industrial robots*. URL: http://www.ifr.org/news/ifr-press-release/the-continuing-success-story-of-industrial-robots-414/ (visited on 09/30/2016).

IFR (2015). *Industrial robot as defined by ISO 8373*. URL: http://www.ifr.org/industrial-robots/ (visited on 09/30/2016).

Ionel, N. (2009). "Agile Software Development Methodologies: An Overview Of The Current State Of Research". *Annals of Faculty of Economics* **4**:1, pp. 381–385.

Ixia (2011). *Deconstructing LTE/4G Testing*. URL: https://support.ixiacom.com/about-us/news-events/corporate-blog/deconstructing-lte4g-testing (visited on 10/01/2016).

Jaap's Puzzle Page (2016). URL: http://www.jaapsch.net/puzzles/cube2.htm (visited on 09/28/2016).

Karmani, R. K., A. Shali, and G. Agha (2009). "Actor frameworks for the JVM platform: a comparative analysis". In: *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*. PPPJ '09. ACM, Calgary, Alberta, Canada, pp. 11–20. ISBN: 978-1-60558-598-7. DOI: 10.1145/1596655.1596658.

Kellerer, H., U. Pferschy, and D. Pisinger (2004). "Introduction to NP-completeness of knapsack problems". In: *Knapsack Problems*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 483–493. ISBN: 978-3-540-24777-7. DOI: 10.1007/978-3-540-24777-7_16.

Kleinrock, L. and F. Tobagi (1975). "Packet switching in radio channels: part I - carrier sense multiple-access modes and their throughput-delay characteristics". *IEEE Transactions on Communications* **23**:12, pp. 1400–1416. ISSN: 0090-6778. DOI: 10.1109/TCOM.1975.1092768.

Lenk, A., M. Klems, J. Nimis, S. Tai, and T. Sandholm (2009). "What's inside the cloud? An architectural map of the cloud landscape". In: *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. CLOUD '09. IEEE Computer Society, Washington, DC, USA, pp. 23–31. ISBN: 978-1-4244-3713-9. DOI: 10.1109/CLOUD.2009.5071529.

LinkLabs (2016). *The Complete List Of Wireless IoT Network Protocols*. URL: http://www.link-labs.com/complete-list-iot-network-protocols/ (visited on 09/28/2016).

Manyika, J., M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon (2015). *Unlocking the potential of the Internet of Things*. URL: http://www.mckinsey.com/business-functions/business-technology/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world (visited on 09/29/2016).

Mattern, F. and C. Floerkemeier (2010). "From active data management to event-based systems and more". In: Sachs, K. et al. (Eds.). Springer-Verlag, Berlin, Heidelberg. Chap. From the Internet of Computers to the Internet of Things, pp. 242–259. ISBN: 3-642-17225-3, 978-3-642-17225-0.

Michieletto, S., S. Ghidoni, E. Pagello, M. Moro, and E. Menegatti (2013). "Why teach robotics using ROS". *Journal of Automation, Mobile Robotics & Intelligent Systems (JAMRIS)*.

Nikoleris, G. (2016). *MMKF15 applied robotics*. Lund University, Sweden.

Normey-Rico, J. E. (2007). *Control of Dead-time Processes*. ISBN: 978-1-84628-829-6.

Persson, P. and O. Angelsmark (2015). "Calvin – merging cloud and iot". *Procedia Computer Science* **52**, pp. 210 –217. ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2015.05.059.

Porter, M. E. (1991). "Towards a dynamic theory of strategy". *Strategic Management Journal* **12**:S2, pp. 95–117. ISSN: 1097-0266. DOI: 10.1002/smj.4250121008.

ROS (2016a). *About ROS*. URL: http://www.ros.org/about-ros/ (visited on 09/28/2016).

ROS (2016b). *ROS Introduction*. URL: http://wiki.ros.org/ROS/Introduction (visited on 09/28/2016).

ROS-Industrial (2016). *ROS History*. URL: http://rosindustrial.org/briefhistory/ (visited on 09/28/2016).

ROS-Industrial (2016). *ROS Industrial description*. URL: http://rosindustrial.org/about/description/ (visited on 09/28/2016).

Sheng, S., Y. Wang, and M. M. Tseng (2009). "Mass customization as a collaborative engineering effort". *International Journal of Collaborative Engineering* **1**:1, pp. 152–167.

Stack Exchange (2016). URL: http://codegolf.stackexchange.com/questions/35002/solve-the-rubiks-pocket-cube (visited on 09/28/2016).

Stivan, G., A. Peruffo, and P. Haller (2015). "Akka.js: towards a portable actor runtime environment". In: *Proceedings of the 5th International Workshop on Programming Based on Actors, Agents, and Decentralized Control*. AGERE! 2015. ACM, Pittsburgh, PA, USA, pp. 57–64. ISBN: 978-1-4503-3901-8. DOI: 10.1145/2824815.2824821.

Teece, D. J. (1993). "The Dynamics of Industrial Capitalism: Perspectives on Alfred Chandler's Scale and Scope". *Journal of Economic Literature* **31**:1, pp. 199–225.

Williamson, O. E. (1981). "The economics of organization: the transaction cost approach". *American Journal of Sociology* **87**:3, pp. 548–577. DOI: 10.1086/227496.

Zühlke, D. Prof. Dr.-Ing. Dr. h.c. (2016). *Industrie 4.0 – vad kan sverige lära?* IVA Syd and SESAM-Sverige. Lund, Sweden.

A

TCP client actor

The following code is the TCP Client actor used as a template to create additional actors for the robot, 5G, and controller clients. The actor has a number of features and a few of them deserve to be given special attention. First of all the ports are defined in the introductory comment. Secondly, the @*manage* statement describes which parameters that should be transferred upon migration. Thirdly, four different actions are defined, *new_connection*, *close_connection*, *receive*, and *send*. Furthermore, the priorities of these actions are defined in the second to last statement. All these actions are preceded by a @*condition* and a @*guard* which are statements that evaluate if an action is ready to fire. Finally, in order to specify the desired requirements, the setup function as well as the requires statement at the end are used. Note that the paths in the requires declarations starts with "calvinsys", which is where capabilities are defined.

```
# -*- coding: utf-8 -*-

# Copyright (c) 2015 Ericsson AB
#
# Licensed under the Apache License, Version 2.0 (the "
    License");
# you may not use this file except in compliance with
    the License.
# You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in
    writing, software
# distributed under the License is distributed on an "
    AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    express or implied.
```

```
# See the License for the specific language governing
    permissions and
# limitations under the License.

from calvin.actor.actor import Actor, ActionResult,
    manage, condition, guard

from calvin.utilities.calvinlogger import get_logger

_log = get_logger(__name__)


class TCPClient(Actor):
    """
    Etablish a TCP connection and forward all tokens.
    Any recevied data on the TCP connection is
        forwarded according to protocol.
    Input:
      data_in : Each received token will be sent out
          through the TCP connection.
      control_in : Each received token will be sent out
          through the TCP connection.
    Output:
      data_out : Data received on the TCP connection
          will be sent as tokens.
    """

    @manage(['address', 'port', 'mode', 'delimiter'])
    def init(self, mode="delimiter", delimiter="\r\n"):
        self.address = None
        self.port = None
        self.EOST_token_received = False
        self.cc = None
        self.mode = mode
        self.delimiter = delimiter
        self.setup()

        if self.address is not None:
            self.connect()

    def connect(self):
        self.cc = self['socket'].connect(self.address,
            self.port, mode=self.mode, delimiter=self.
```

```
        delimiter)

    def will_migrate(self):
        if self.cc:
            self.cc.disconnect()

    def did_migrate(self):
        self.setup()
        if self.address is not None:
            self.connect()

    def setup(self):
        self.use('calvinsys.network.socketclienthandler
            ', shorthand='socket')
        self.use('calvinsys.native.python-re',
            shorthand='regexp')

    @condition(action_input=['data_in'])
    @guard(lambda self, token: self.cc and self.cc.
        is_connected())
    def send(self, token):
        self.cc.send(token)
        return ActionResult(production=())

    @condition(action_output=['data_out'])
    @guard(lambda self: self.cc and self.cc.
        is_connected() and self.cc.have_data())
    def receive(self):
        data = self.cc.get_data()
        return ActionResult(production=(data,))

    # URI parsing - 0: protocol, 1: address, 2: :port
    URI_REGEXP = r'([^:]+)://([^/:]*)(:[0-9]+)'

    @condition(action_input=['control_in'])
    @guard(lambda self, control: control['control'] ==
        'connect' and not self.cc)
    def new_connection(self, control):
        uri = self['regexp'].findall(self.URI_REGEXP,
            control['uri'])
        uri_parts = uri[0]
        protocol = uri_parts[0]
```

```
        if protocol != 'tcp':
            _log.warn("Protocol␣'%s'␣not␣suuported" % (
                protocol,))
        else:
            self.address = uri_parts[1]
            self.port = int(uri_parts[2][1:])
            self.connect()

        return ActionResult(production=())

    @condition(action_input=['control_in'])
    @guard(lambda self, control: control['control'] ==
        'disconnect' and self.cc)
    def close_connection(self, control):
        self.cc.disconnect()
        self.cc = None
        return ActionResult(production=())

    def exception_handler(self, action, args, context):
        """Handler ExceptionTokens"""
        self.EOST_token_received = True
        return ActionResult(production=())

    action_priority = (new_connection, close_connection
        , receive, send)
    requires = ['calvinsys.network.socketclienthandler'
        , 'calvinsys.native.python-re']
```

B

Socket client handler

Similarly to the TCP client actor, a socket client handler has been used as a template. As mentioned in the previous chapter the TCP client actor has requirements in order to run, and that requirement is the socket client handler capability. As seen in the code, the socket client handler provides functions concerned with the socket communication. The socket client handler shown is an example of how a capability is implemented in Calvin.

```
# -*- coding: utf-8 -*-

# Copyright (c) 2015 Ericsson AB
#
# Licensed under the Apache License, Version 2.0 (the "
    License");
# you may not use this file except in compliance with
    the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in
    writing, software
# distributed under the License is distributed on an "
    AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    express or implied.
# See the License for the specific language governing
    permissions and
# limitations under the License.

from calvin.runtime.south.plugins.async import
    client_connection
```

```
from calvin.utilities.calvin_callback import CalvinCB
from calvin.utilities.calvinlogger import get_logger

_log = get_logger(__name__)


class ClientHandler(object):
    def __init__(self, node, actor):
        _log.debug("Client handler %s for %s" % (self,
            actor.id))
        self._actor = actor
        self._node = node
        self._connections = {}

    def _trigger_sched(self):
        self._node.sched.trigger_loop(actor_ids=[self.
            _actor.id])

    # Callbacks from socket client imp
    def _disconnected(self, handle, addr, reason):
        self._connections[handle]['connection'] = None
        self._push_control(("disconnected", addr,
            reason), handle)

    def _connected(self, handle, connection_factory,
        addr):
        self._connections[handle]['connection'] =
            connection_factory
        self._push_control(("connected", addr, ""),
            handle)

    def _connection_failed(self, handle,
        connection_factory, addr, reason):
        self._push_control(("connection_failed", reason
            ), handle)

    def _push_data(self, handle, data):
        self._connections[handle]['data'].append(data)
        self._trigger_sched()

    def _push_control(self, data, handle):
        self._connections[handle]['control'].append(
            data)
```

```
        self._trigger_sched()

    # External API
    def connect(self, addr, port, mode="raw",
        connection_type="TCP", delimiter="\r\n"):
        handle = "%s:%s:%s" % (self._actor.id, addr,
            port)

        if connection_type == "TCP":
            connection_factory = client_connection.
                TCPClientProtocolFactory(mode=mode,
                delimiter=delimiter, callbacks={'
                data_received':[CalvinCB(self.
                _push_data, handle)]})
        elif connection_type == "UDP":
            connection_factory = client_connection.
                UDPClientProtocolFactory(callbacks={'
                data_received': [CalvinCB(self.
                _push_data, handle)]})

        connection_factory.callback_register('connected
            ', CalvinCB(self._connected, handle,
            connection_factory))
        connection_factory.callback_register('
            disconnected', CalvinCB(self._disconnected,
             handle))
        connection_factory.callback_register('
            connection_failed', CalvinCB(self.
            _connection_failed, handle,
            connection_factory))
        connection_factory.connect(addr, port)
        self._connections[handle] = {'data': [], '
            control': [], 'connection': None}
        return ClientConnection(self, handle)

    def disconnect(self, handle):
        self._connections[handle]['connection'].stop()

    def get_data(self, handle):
        return self._connections[handle]['data'].pop()

    def is_connected(self, handle):
```

```
            return self._connections[handle]['connection']
                is not None

        def have_data(self, handle):
            return len(self._connections[handle]['data'])

        def send(self, handle, data):
            return self._connections[handle]['connection'].
                send(data)

        def have_control(self, handle):
            return len(self._connections[handle]['control'
                ])

        def get_control(self, handle):
            return self._connections[handle]['control'].pop
                ()


    class ClientConnection(object):
        def __init__(self, client_handler, handle):
            self._client_handler = client_handler
            self._handle = handle

        def _call(self, func, *args, **kwargs):
            return func(self._handle, *args, **kwargs)

        def __getattr__(self, name):
            class Caller(object):
                def __init__(self, f, func):
                    self.f = f
                    self.func = func

                def __call__(self, *args, **kwargs):
                    return self.f(self.func, *args, **
                        kwargs)

            if hasattr(self._client_handler, name):
                if callable(getattr(self._client_handler,
                    name)):
                    return Caller(self._call, getattr(self.
                        _client_handler, name))
                else:
```

```
                return getattr(self._obj, name)

        else:
            # Default behaviour
            raise AttributeError


def register(node, actor):
    """
        Called when the system object is first created.
    """
    return ClientHandler(node, actor)
```