NTNU
Norwegian University of
Science and Technology

Subsea Communication

Implementing and Evaluating Protocols

Sondre Kyrkjeteig

Master of Science in Cybernetics and Robotics
Submission date: June 2016
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Assignment Text

Thesis Title: Subsea Communication - *Implementing and Evaluating Protocols*

Background

Within a subsea control system various industrial Ethernet protocols are implemented for process data exchange on the link between topside & subsea. Also, different types of instrumentation networks are used to collect data from sensors & intelligent devices at the seabed. In this project different protocols & networks shall be evaluated based on criterion from OneSubsea Processing. As a minimum the following protocols shall be evaluated: Modbus TCP, Ethernet/IP, and OPC UA for topside - subsea communication. CAN bus (SIIS level II) and IWIS (API 17F) shall be evaluated for collecting sensor & actuator data to the SCM. The project shall also consist of a practical implementation of industrial protocols.

Work Description

1. Give an introduction and perform an evaluation of the topside protocols and the subsea instrumentation networks that are mentioned above. Other protocols shall be evaluated if they are found applicable.
2. Look into the standardization work performed within subsea control systems, and give a brief theoretical description on how subsea control and communication is performed in a general subsea project.
3. Implement the Modbus TCP application protocol for message exchange between micro-controllers.
4. Investigate and find a suitable OPC UA solution for a subsea communication network. Implement this OPC UA solution on an micro-controller/computer.

Start date: January 11, 2016 **Due date:** June 06, 2016

Supervisor:	Tor Onshus
Co-advisor:	Stian Hjellvik Askeland (OneSubsea Processing)

Preface

This master thesis is a result of my work in TTK4900 at the Department of Engineering Cybernetics during the spring semester of 2016. The master thesis is a compulsory part of the 2-year program for achieving a Master of Science (MSc) degree in Cybernetics and Robotics at NTNU. This master thesis is not a direct continuation on previous thesis/project work, all work related to this project started up in the middle of January 2016. The project is carried out in co-operation with OneSubsea Processing in Bergen. The department in Bergen is a provider of subsea production systems by use of equipment such as meters, pumps, and wet gas compressors. They also engineer swivels. Their main goal is to provide lift and boosting of hydrocarbons. The idea of the master thesis was brought up by myself, Øyvind Reksten, and Frank Midtveit at OneSubsea Processing.

This master thesis has been quite informative, and given me the possibility of learning more about a topic I find very interesting. Especially, the insight into subsea standardization work has been quite valuable. Its made me able to understand why system solutions we have today exists. The theoretical work has consisted of reading and understanding communication standards, recommended practices, user manuals, and research papers. The practical part has consisted of implementing two different communication protocols that can be used for the same purpose. The first task was reading and interpreting the official Modbus TCP specification and implementing this for message exchange between two microcontrollers. The second task involved implementation of OPC UA by use of a software development kit.

My experience of reading standards is that many of them leaves room for interpretation, and a recommended practice is quite helpful for understanding the practical concepts and details. This master thesis is intended for readers that are familiar with Ethernet, subsea control systems, and different types of communication protocols.

Trondheim, 2016-06-03

Sondre Kyrkjeteig

Acknowledgment

First of all I would like to thank my supervisor at NTNU, Professor Tor Onshus. Thank your for providing me with guidance and support during the whole master thesis. Answers on questions related to technical issues, progress plan, protocol clarifications, and document details are highly appreciated.

I would like to thank Øyvind Reksten and Frank Midtveit at OneSubsea for their willingness and efforts in composing an assignment text for a master thesis that sounded interesting on beforehand, and has been informative during the project phase. I would also like to thank my co-advisor at OneSubsea, Stian Hjellvik Askeland. Thank you for providing me with inspiration, material, and answers to my detailed questions.

S.K.

Sammendrag og konklusjon

Interessen for å gjennomføre dette prosjektet er for å evaluere de ulike protokollene og instrumenteringsnettverkene som ofte brukes i undervannskontrollsystem. Dette arbeidet skal utføres ved å peke ut hovedforskjeller, illustrere prinsipper, og sammenligne ytelse & den praktiske implementasjonserfaring av relevante protokoller. Modbus TCP og Ethernet/IP er to globalt anerkjente og åpne protokoller som blir brukt for å utveksle prosessdata mellom plattformen og undervannsutstyret. OPC UA derimot er en relativ ny kommunikasjonsprotokoll som vanligvis blir brukt for å utveksle prosessdata mellom systemer fra ulike leverandører. UA er inkludert i dette prosjektet som et forslag til å utføre fremtidig datautveksling mellom plattformen og undervannsutstyret. Dette arbeidet blir gjort ved å foreta en teoretisk evaluering og praktisk implementasjon av UA. Den praktiske implementasjonen utforsker muligheten for å implementere en fremtidig OPC UA løsning i et kommunikasjonsnettverk for undervannssystem. Målet er at resultatet fra denne masteroppgaven kan brukes for å argumentere hvilke industriell Ethernet protokoll som er best egnet for linken mellom plattformen og undervannsutstyret basert på de tekniske egenskapene.

Ett av de opprinnelige målene til masteroppgaven var å implementere de spesifikke protokollene fra oppgaveteksten på identisk maskinvare og nettverk. Deretter skulle en utføre en detaljert ytelsessammenligning for å støtte opp under eventuelle ulikheter som ble avdekket i den teoretiske sammenligningen. På grunn av urimelig høy kostnad for to stk FPGA utviklingsverktøy som inkluderte de nødvendige protokollene og programvaren så ble falt dette målet bort. Målene til masteroppgaven ble derfor endret til å involvere en praktisk del som gikk ut på å finne passende maskinvare, beskrive, sammenligne, og utføre den praktiske implementasjon av to ulike protokoller. Den teoretiske biten har bestått i å forklare undervannskommunikasjon, og trekke frem hovedforskjellene mellom de industrielle Ethernet protokollene som er nevnt i oppgaveteksten. En evaluering av de ulike standardene for utveksling av sensor & aktuator data på havbunnen er også en del av omfanget i denne master oppgaven.

Den teoretiske delen av denne rapporten evaluerer og sammenligner Modbus TCP, Ethernet/IP

og OPC UA protokollene basert på egenskaper og studier om ytelsen. Evalueringen konkluderer med at kommunikasjonsprotokoller som er basert på standard TCP/IP og Ethernet er for løst definert til å kunne spesifisere ytelsen. Ytelsen vil rett og slett variere for mye i hvert enkelt tilfelle i forhold til å kunne tallfeste spesifikke ytelsesverdier. Maskinvare, nettverk, og effektiviteten på meldingsutvekslingene i applikasjonslaget vil påvirke ytelsesverdiene i stor grad. Ethernet/IP er den eneste av av de tre protokollene som spesifiserer oppnåelige tallfestede ytelsesverdier i offisielle standarder. Studier avslører at Modbus TCP ikke har en passende kommunikasjonsmodell for å utveksle sanntidsdata, og er heller ikke ment å ha spesifiserte ytelsesverdier oppgitt i standarder pga. sitt brede bruksområder. Det er ikke mulig å tallfeste ytelsen til OPC UA pga. at det er en teknologiuavhengig protokoll (kun definert for de øvre lagene) med et rikt utvalg av service mekanismer som kan aktiveres/deaktiveres. Uansett så er beskrivelser av artikler som ser nærmere på ytelsespåvirkningen av: sikkerhetsmekanismer, aktive servicer, nettverkstrafikk, utviklingsverktøy, og et økende antall klienter for OPC UA en del av denne masteroppgaven. Sammenligningen mellom Modbus TCP og Ethernet/IP konkluderer med at Ethernet/IP har bedre sanntidsegenskaper, bedre multisending, og bedre støtte i forhold til egenskaper som tidssynkronisering & sikkerhetsprotokoll. Men Modbus TCP garanterer for data integriteten siden den bruker TCP som transport protokoll, i motsetning til Ethernet/IP som bruker UDP. OPC UA er betraktet som et interessant valg for fremtidig implementering pga. solid støtte for å modellere kompleks arkitektur, og det rike utvalget av service mekanismer som er beskrevet i UA spesifikasjonen. Derimot så er ikke OPC UA en sanntidsprotokoll, og er ikke egnet for sikkerhetskritiske system.

Introduksjonen og evalueringen av CAN (SIIS level II) og IWIS for å utveksle data på sensornivå konkluderer med at en direkte sammenligning mellom de to ulike prinsippene er utfordrende pga. at de er konstruert for to ulike formål. CAN standarden beskriver hvordan instrumenteringsnettverket som blir brukt for å utveksle data mellom undervannsutstyr på havbunnen skal konstrueres. IWIS definerer et grensesnitt som kan brukes av intelligent måleutstyr som er levert av en tredjepart. Ved å bruke IWIS grensesnittet så kan deres utstyr koble seg opp mot undervannskontrollsystemet. Undervannskontrollsystemet vil da fungere som en transparent transport funksjon mellom havbunnen og plattformen. Evalueringsseksjonen for denne delen

diskuterer relevansen av å bruke andre sensornettverk enn CAN, utfordringer relatert til hastigheten og det fysiske laget til IWIS grensesnittet, og trekker frem implementeringserfaring fra industrien.

Den praktiske delen av prosjektet bestod av en implementering av to ulike kommunikasjonsprotokoller som kan bli brukt til det samme oppsettet i et kommunikasjonsnettverk for undervannssystemer. Løsningene er diskutert, og sammenlignet basert på den praktiske erfaringen som ble oppnådd ved implementasjon. Den første praktiske implementasjonen bestod i å studere og tolke Modbus TCP spesifikasjonen, og implementere protokollfunksjonene i et applikasjonslag. Et brukerlag ble så senere lagt til for å demonstrere meldingsutvekslingen mellom to mikrokontrollere. Den andre delen bestod i en praktisk implementasjon av OPC UA kommunikasjonsprotokollen på en mikrodatamaskin (Raspberry Pi). Denne implementasjonen ble gjennomført ved å bruke et programvareutviklingsverktøy som tilbyr UA servicer iht. den offisielle OPC UA spesifikasjonen.

Den praktiske delen har resultert i to fungerende implementasjoner som er vedlagt denne masteroppgaven. Implementasjonen av Modbus TCP applikasjonslaget er operativt for meldingsutveksling mellom to mikrokontrollere, en klient og en server. Modbus TCP implementasjonen gikk forholdsvis greit ettersom Modbus spesifikasjonen er oversiktlig og detaljert. Det er lett å forstå hvorfor bruken av Modbus TCP er så utbredt i industrien. Men Modbus TCP er en forbindelsesbasert forespørsel/reaksjons-protokoll som ikke innehar de beste sanntidsegenskaper. En annen ulempe med Modbus TCP sammenlignet med Ethernet/IP og OPC UA er manglende støtte for objekt-modellering. Rent praktisk vil dette bety at klienten må vite registeradressene til den respektive serveren når den skal sende forespørsel til serveren(e). OPC UA løsningen i denne masteroppgaven fremstår som et eksempel på en fremtidig implementasjon av OPC UA i et kommunikasjonsnettverk for undervannsutstyr. Løsningen demonstrerer hvordan en skal implementere sensor & aktuator data i adresseområdet til en UA server ved å definere objektmodeller som representerer fysisk data. Hele den praktiske OPC UA implementasjonen er utført ganske så ulikt Modbus implementasjonen. OPC UA spesifikasjonen beskriver metoder og servicer for å modellere kompleks arkitektur, den spesifiserer ikke hvilke teknologi en skal bruke. En

OPC UA implementasjon er vanligvis utført vha. et omfattende programvareutviklingsverktøy som støtter tjenestene i den offisielle UA spesifikasjonen som er definert av OPC Foundation. Implementasjonsutfordringen for OPC UA var å definere objekt-modeller ved å bruke riktig UA terminologi & modelleringsverktøy, og bruke riktig type services & managers i programvareutviklingsverktøyet.OPC UA løsningen er implementert med støtte for Data Access (DA), Historical Data Access (HDA), og Alarms & Conditions (AC) for å demonstrere skalerbarheten til OPC UA. Sammenlignet med Modbus TCP så er OPC UA kommunikasjonen mer abstrakt, den trenger mer ressurser, og den har dårligst effektivitet blant de to. Men den støtter forskjellige typer meldingsutveksling (forespørsel/reaksjon, Subscription (abonnement), server-server). OPC UA løsningen er også mer kompleks, og det trengs mer arbeid for førstegangsimplementasjon av teknologien. UA er derimot mye mer fleksibel når grunnstammen er på plass. De fleste parametere kan da endres, og flere OPC UA variabler kan da enkelt legges til. Den største ulikheten mellom de to praktiske implementasjonene var detaljnivået. Ved implementasjonen av Modbus TCP var fokuset på å få bits & bytes"i meldingsstrukturen riktig, mens for OPC UA var utfordringen å definere objekt-modeller og implementere disse i utviklingsverktøyet ved å bruke de riktige forhåndsdefinerte servicene.

Summary and Conclusion

The interest for carrying out this project is to evaluate the various protocols & instrumentation networks used in subsea control systems. This work shall be performed by pointing out the main differences, illustrate principles, and compare the performance & practical implementation experience of subsea relevant protocols. Modbus TCP and Ethernet/IP are two globally accepted and open protocols used for process data exchange between topside and subsea. OPC UA is on the other hand a relatively new communication protocol which is typically used for exchanging process data between systems from different vendors. UA is included in this project as a suggestion to perform prospective data exchange between topside and subsea. Thus, a theoretical evaluation and practical implementation of UA is performed. The practical implementation investigates the possibility of implementing a prospective OPC UA solution in a subsea network. The aim is that results from this thesis can be used to argue which industrial Ethernet protocol is best suited for topside communication based on the technical properties.

One of the initial objectives in this project was to implement the specific protocols from the assignment text on the same hardware & network, and then perform a detailed benchmark comparison to back potential dissimilarities from the theoretical analysis. However, due to unreasonable cost for two FPGA development kits including all necessary protocols and software this was omitted. The objectives of the master thesis therefore shifted focus towards a practical part that involved finding suitable hardware, describe, compare, and carry out the practical implementation of two different protocols. The theoretical part has consisted of explaining the subsea communication, and describe & highlight the differences between industrial Ethernet protocols used for topside communication. Also, an evaluation of the various standards for exchanging sensor & actuator data at the seabed are within the scope of this thesis.

The theoretical part of this thesis evaluates and compares the Modbus TCP, Ethernet/IP, and OPC UA protocols based on features, and performance studies. The theoretical evaluation concludes that communication protocols based on the TCP/IP communication stack and Ethernet standard are to loosely defined in order to specify their performance. The performance will simply vary to much in each single case in order to quantify specific performance indicators. Hardware, network type, and the actual exchange management efficiency in the application layer will influence the performance quantities in great extent. Ethernet/IP is the only protocol of the three that quantifies indicators for obtainable performance in official standards. Studies reveals that Modbus TCP does not have a suited communication model for real-time data exchange, and its not supposed to have performance indicators due to its wide range of application. The performance quantification of OPC UA is impossible because its a technology independent protocol (only defined at higher layers) with a wide selection of service mechanisms that can be enabled/disabled. However, further details explaining the research studies on performance impact of: the security mechanisms, enabled services, network traffic, software development kits, and the increasing number of clients for OPC UA are described in this master thesis. The comparison of Modbus TCP and Ethernet/IP concludes that, Ethernet/IP has better real-time properties, better multicast performance, and better support for features like time synchronization & safety protocol. However, Modbus TCP guarantees for the data integrity due to the use of TCP as transport protocol in contrast with Ethernet/IP which uses UDP. OPC UA is considered to be an interesting choice for prospective implementation due to its support for modeling complex architecture, and its wide choice of service mechanisms as defined in the UA specification. OPC UA however is not a real-time protocol and its not suited for safety critical systems.

The introduction and evaluation of CAN bus (SIIS level II) & IWIS for data exchange at sensor level, concludes that a direct comparison of the two principles are challenging because they are engineered for two different purposes. The CAN bus (SIIS level II) describes the extensive subsea standard for instrumentation network used to exchange data with sensor and actuators at the seabed. IWIS defines an interface that third-party intelligent well equipment (downhole measurement tools) uses to integrate into a subsea production control system. The subsea control system will then provide a transparent transport function between subsea and topside. The evaluation section discusses the relevance of using another type of sensor network than CAN bus, challenges related to to the baud-rate and physical layer of the IWIS interface, and highlights the implementation experience from the industry.

The practical part of this thesis consisted of implementing two different communication protocols that can be used for the same setup in a subsea communication network. The solutions are discussed, and compared based on the practical experience gained by the implementation. The first practical implementation was to study and interpret the Modbus TCP specification, and implement the protocol functions in an application layer. A user-defined layer was later added to demonstrate the exchange of message functions between two microcontrollers. The second part was a practical implementation of the OPC UA communication protocol on a microcomputer (Raspberry Pi). This implementation was performed using a software development kit that provides the official UA services as defined in the OPC UA specification.

The practical part has resulted in two fully operational implementations that are attached to this master thesis as appendices. The Modbus TCP application-layer implementation is fully operational for message exchange between two microcontrollers, one client and one server. Due to a clear and detailed specification, the Modbus TCP implementation experience was quite positive. Its easy to understand why its usage is so widespread in the industry. However, its a connection-based request/response protocol that does not hold the best real-time properties. Also, a disadvantage of Modbus TCP compared to Ethernet/IP and OPC UA is the lack of support for object modelling. This means the client must know the register addresses at the server(s) in order to request the correct data. The OPC UA solution in this thesis serves as an example for prospective implementation of OPC UA in a subsea communication network. The solution demonstrates how to implement sensor & actuator data in the address space on an UA server by defining object models that represent real-world data. The practical OPC UA implementation is performed quite different than for Modbus. The OPC UA specification describes methods & services to model complex architecture, it does not specify what type of technology to use. An OPC UA implementation is normally based on a comprehensive software development kit that supports the services in the official UA specification defined by the OPC Foundation. The implementation challenge for OPC UA has involved defining object models by use of correct UA terminology & modelling tools, and also use the correct services & managers in the software development kit. The solution is implemented with support for Data Access (DA), Historical Data Access (HDA), and Alarms & Conditions (AC) to demonstrate the scalability of

OPC UA. Compared to Modbus TCP, the OPC UA communication is more abstract, it requires more resources, and it has the worst efficiency of the two protocols. However, it supports various types of message exchange (request-response, subscription, server-server). The OPC UA solution is more complex, and the initial implementation of the technology demands more efforts. On the other hand OPC UA is much more flexible when the basic framework is in place. Most of the parameters can then be changed, and several OPC UA tags can easily be added. The greatest dissimilarity of the two practical implementations were the degree of details. The implementation of Modbus TCP for an embedded system focused on getting the "bits & bytes" of the message structure correct. While for OPC UA the challenge was defining object models and implementing these in the SDK using the predefined services.

Abbreviations

ADU	Application Data Unit
CAN	Controller Area Network
CiA	CAN in Automation
CIP	Common Industrial Protocol
DCS	Distributed Control System
FPGA	Field-Programmable Gate Array
FPSO	Floating Production, Storage and Offloading unit
ICSS	Integrated Control and Safety System
IWE	Intelligent Well Equipment
IWIS	Intelligent Well Interface Standard
JIP	Joint Industry Project
MBAP	Modbus Application Protocol
MCS	Master Control System
PDU	Protocol Data Unit
RPi	Raspberry Pi
RTE	Real Time Ethernet
SCM	Subsea Control Module
SDK	Software Development Kit
SEM	Subsea Electronic Module
SIIS	Subsea Instrumentation Interface Standardization
SPCU	Subsea Power and Communication Unit

Terminology

Address space	Collection of information an OPC UA server makes visible to its clients [28]
Bandwidth	A measure on how fast data can potentially be sent over between nodes
Cycle time	The communication time required by the controller to both collect and update the data memories of all sensors and actuators [36]
Determinism	The ability of the communication protocol to guarantee that a message is sent or received in a finite and predictable amount of time [2]
Multicast	Describes a one-to-many communication scenario, e.g. one client and several servers
Octet	A sequence of eight bits (a byte)
Payload	A measure on how much of the message data that is actually useful for the user, when you exclude headers and checksums
Real-time	The ability of a system to provide required data in a bounded time
Throughput	A measure on the actual amount of transmitted data between nodes

Contents

Chapter 1

Introduction

1.1 Background

The assignment text for this master thesis was composed in co-operation with OneSubsea Processing. They support a wide range of industrial communication protocols for their control systems, the criterion for selecting a specific communication protocol is not necessarily always performed from a technical perspective. The reference manual [9] claims that the choice of industrial protocol(s) typically are based on which protocols a specific controller supports, and what the supplier can deliver of equipment & provide technical support for. Customer requirements, historical, economical, and personal (experience, training) reasons could play just an important role as the best match from a theoretical perspective. Regardless, the interest for performing this thesis is to provide technical theoretical knowledge about subsea communication by illustrate principles, highlight differences, compare the industrial Ethernet protocols, and look into differences for subsea instrumentation networks. Hopefully can the result of this master thesis provide basic introduction to subsea communication systems, and recommend best suited protocols from a technical perspective. The thesis will also review some of the work performed on subsea standardization.

This thesis also looks into concepts such as event/time-triggered communication, and the requirements for a safety protocol. A subsea control system that is implemented outside the platform's distributed control system, and does not have safety functions can avoid strict requirements for usage of a safety protocol. Further details regarding regulations and the mechanisms to achieve a safety protocols is discussed in Section 3.5.3. However, in this thesis its assumed that during normal operation the only data exchange between topside and subsea is related to process control messages. Some subsea suppliers enables solutions with prioritization of service messages and production control messages, so that service/updates can be performed in parallel with the production of hydrocarbons. The possibility of message prioritization is commented in the section related to topside communication.

Over the years, the oil & gas industry has steered towards the use of an industrial Ethernet protocol on the link between topside and subsea, and today this has more or less become the standard solution. Industrial Ethernet is well suited for this purpose since its physical layer supports both twisted pair cables and fiber optics. The distance tends to be ten's of kilometers, and the process data exchange therefore benefits from using a protocol that supports high data rate. At the seabed, sensor-level communication is implemented to allow frequent exchange of small data packets between subsea instruments or intelligent well devices. Within subsea control systems there are mainly two specifications for this purpose, the CAN bus (SIIS level II) and IWIS. Both of these specifications are introduced and evaluated in Section 4.

Detailed information regarding the Ethernet standard, the TCP/IP communication stack, the CANopen standard or the subsea standard (ISO-13628) is not within the scope of this thesis. Complementary information is cited if its found relevant. The requirements for communication protocols used in a subsea control system is specified in the ISO-13628 (explained in more details later). The main focus in this thesis will be to sort out differences between the given protocols, and not suggest the usage of other protocols.

This project contains two independent practical implementations of communication protocols. First Modbus TCP is implemented for message exchange on two microcontrollers. Compared to a subsea communication network, one of the microcontrollers represents the client (topside controller) and the other microcontroller represents the server (embedded system installed at the seabed). The other implementation is the scalable OPC UA protocol. OPC UA is implemented to illustrate how a prospective implementation of OPC UA can be performed in a subsea communication network. The solution demonstrates how to implement subsea instrumentation data in the UA address space by defining object models that represent real world data. The UA implementation represents the same server-client communication model as Modbus TCP. A Raspberry Pi represents the server, and a host computer represents the client. The implementation of OPC UA on a microcomputer is an interesting topic not only to subsea control systems, but also for use in Internet of Things (IoT).

1.2 Literature Survey

There are several articles discussing the various types of industrial/real-time Ethernet protocols. Some examples are the industry prospective on real-time Ethernet [16], and the minimum cycle time analysis of Ethernet-based real-time protocols [36]. Detailed performance studies on industrial/real-time Ethernet protocols executed by an independent third-party has however been challenging to find.

Several articles related to the OPC UA performance has been published. The relevant articles considers the performance influence for the CPU load and network traffic due to increasing number of clients & monitored items [17]. The performance impact due to the security mechanisms are discussed in [11], and the performance impact of the subscription mechanism & transport protocols are discussed in [10]. OPC UA is a relatively new communication protocol where several articles are related to how an OPC UA implementation should be performed for various communication scenarios, and what type of service mechanisms should be exploited. The article discussing OPC UA based solutions for offshore integrated operations [37] is relevant for this thesis. It debates whether or not OPC UA has secure communication, secure information sources and standardized information exchange. The article [37] also refers to research related to implementation of OPC UA in field equipment for process automation.

Information related to IWIS and SIIS has mainly been found in standards [4, 20] and recommended practices [30, 31]. The article [13] describing the experience from implementation of IWIS and SIIS in subsea control systems are also relevant.

1.3 Objectives

The main objectives of this Master's project are

1. Give an introduction and perform an evaluation of Modbus TCP & Ethernet/IP for topside - subsea communication. Look into the possibilities of using OPC UA for prospective topside - subsea communication.

2. Give a brief description on how subsea control and communication is performed in a general subsea production system. Explain some of the standardization work performed within subsea control systems. Briefly discuss the differences on event-triggered and time-triggered communication.

3. Give an introduction and evaluate the CAN bus (SIIS level II) and IWIS for collecting subsea instrumentation data.

4. Implement the Modbus TCP application protocol for message exchange between micro-controllers.

5. Investigate and find a suitable OPC UA solution for a subsea communication network. Implement this OPC UA solution on a micro-controller/computer.

1.4 Structure of the Report

Chapter 2: Provides a brief introduction: to the main subsea components, relevant standards, reasons for using industrial Ethernet protocols for topside communication, and the principles of event/time-triggered communication.

Chapter 3: Theory related to the various industrial Ethernet protocols used for topside communication are presented in this chapter.

Chapter 4: Describes and evaluates the specifications for subsea sensors & intelligent well devices.

Chapter 5: Documents the practical implementation of the Modbus TCP application layer protocol for message exchange between microcontrollers, and an OPC UA server on a microcomputer.

Chapter 6: The final chapter contains the discussion and provides recommendations for further work.

Chapter 2

Subsea Introduction

2.1 Intro

The subsea control system is part of the infrastructure to control the production of oil & gas. It operates valves & chokes, and collects process data regarding temperature, pressure, flow, sand detection, etc. The systems ranges in complexity from a single well linked to a fixed platform, FPSO, or onshore refinery, to several wells clustered with help of a manifold. This thesis focuses primarily on topside communication and subsea instrumentation network, but a brief introduction of the most relevant equipment is given in this chapter. The subsea control module (SCM) and the subsea electronic module (SEM) are considered to be highly relevant and therefore given a more detailed explanation. More specific details regarding subsea production system principles, advantages, disadvantages, limitations, and characteristics are found at [7, 20, 4].

Figure 2.1 is an overview example of a subsea production system. At topside there are systems providing power, hydraulic, communication signals & logic, and chemicals. Power, hydraulic, chemicals, and communication signals are transported via the umbilical from topside to the subsea equipment located at the seabed. Communication signals are normally transferred by use of fiber optics, but a copper link may also be used. Before the introduction of fiber link, powerline communication was the main method to transfer binary data via the power signals. The umbilical is terminated and connected to subsea equipment, such as subsea trees, pumps, and compressor. Figure 2.2 illustrates a more direct & simplified overview of a subsea control

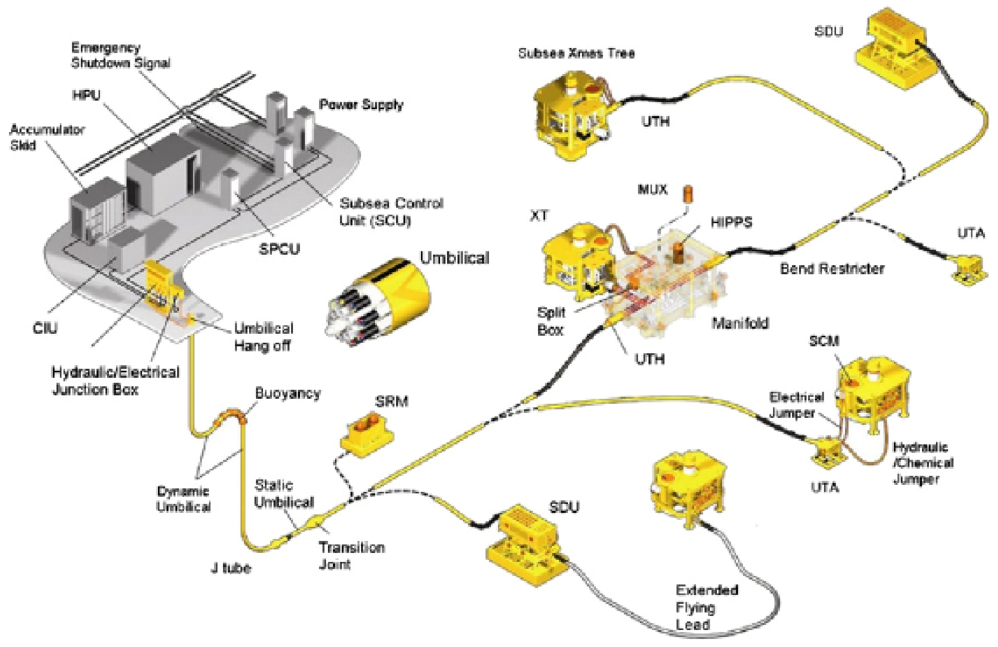


Figure 2.1: Subsea production system overview [7, Ch. 7]

system which is relevant for this project. This master thesis discusses protocols used for collecting subsea instrumentation data at the SEM, and industrial Ethernet protocols used for data exchange between the MCS and the SEM. Two key components in the control system are the master control station (MCS) and distributed control station (DCS). The master control station is the unit that control and monitor the subsea production system [20]. The distributed control system is the platform's decentralized control system. The subsea power and control unit (SPCU) connects to the subsea control modules (SCMs) and provides electrical power to subsea control equipment. The SPCU is also used to obtain communication between topside and subsea by use fiber modems. Filters, and adjustable power are also requirements for the SPCU [7].

2.2 Subsea Control Module

The subsea control module (SCM) is a independently retrievable unit used to provide well functions during the production phase of subsea oil and gas production. The module is normally installed directly onto the equipment to be controlled, such as the subsea tree, pump, or compressor. The SCM is designed to be installed or retrieved by using a single lift wire and some

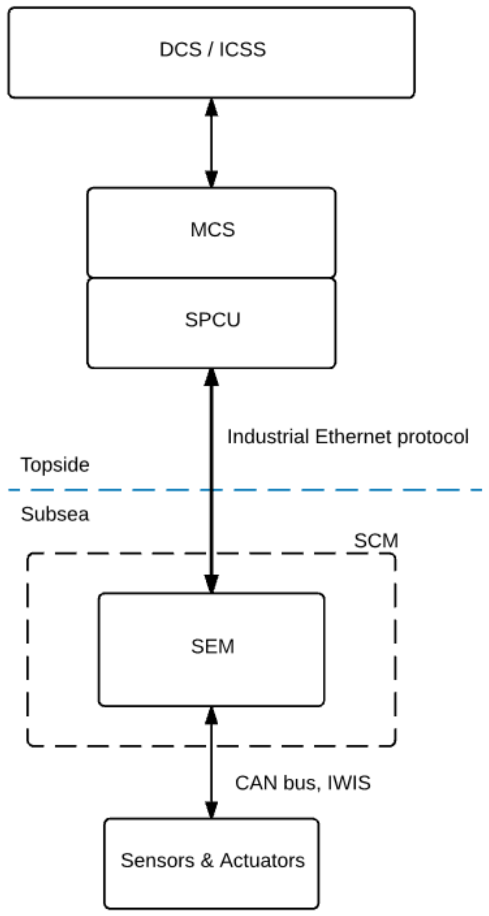


Figure 2.2: Subsea control system flow overview

remotely operated vehicle (ROV) assistance. The SCM receives the hydraulics, electrical power, and communication signals from topside. Typical SCM functions are

- To communicate with and transfer data between the topside controller and the subsea instrumentation
- Monitoring of the system (pressure, temperature, flow, sand detection)
- Actuation of fail-safe return at production tree actuators and downhole safety valves.
- General valve operations as requested by the topside controller

A short distance between the SCM and the subsea production results in benefits like quicker valve response, better valve control (less risk), continuous situation report of the system, and reduces umbilical weight & costs. The physical external SCM housing has hydraulic and electrical connectors, ROV handled connectors, and a stab plate for connectors. The inside consist of a hydraulic system (including accumulators), a lock down mechanism, one or several subsea electronic modules (SEM), ambient pressure measurement, and water leak detection. The SCM housing is normally filled with a dielectric fluid and pressure compensated to provide a secondary barrier to ingress from sea water [7, 20, 4].

2.3 Subsea Electronic Module

The subsea electronic module (SEM) is the module that collects subsea process data & controls subsea actuators. The SEM is mounted inside the SCM, and is connected to the power and communication coming from topside. The SEM is implemented to allow control and monitoring of subsea instrumentation. A SEM tends to consist of the follow items:

- Subsea control electronics and data storage
- I/O and communication modules
- Subsea signal modem
- Power supply module with power management system
- Electrical connectors/penetrators and cables

The SEM is a embedded system which acts as a local control unit, a temporary storage device, and a gateway between the industrial Ethernet network (topside communication) and the subsea process data which are available through 4-20 mA, CAN bus, or Ethernet. The SEM software must be implemented with support for housekeeping data (temperature, humidity, pressure, voltage & current, auxiliary attributes) [7, 20, 4].

2.4 Subsea Control Systems Standard

ISO 13628 is the standard for *design and operation of subsea production systems* within the petroleum industry. Part 6 of the standard regards the *subsea production control systems*. This part is applicable to the architecture, design, fabrication, installation, testing, and operation of subsea production control systems. Part 6 is also essential for the general functionality of subsea control systems, and in particular the execution of subsea communication. Further requirements for the SCM and the SEM are specified in the standard.

For communication between topside and subsea the ISO 13628-6 does not specify which communication protocol that must be used. Some of the requirements are high reliability and sufficient capacity to handle the required traffic in all foreseeable situations. Its also a requirement that the communication is based on proven design or an industry standard. More information on suitable communication protocols for subsea is found at [20, 7.4.6]. More information on why Ethernet is a match can be found in Section 2.6. Two various solutions for collecting subsea instrumentation data are standardized in the ISO 13628-6: CAN bus (SIIS level II) and IWIS, both of these solutions are presented later in Chapter 4.

2.5 OSI Reference Model

The open system interconnect (OSI) reference model is used as a common reference for development of data communication standards. It works as a structural aid to understand how the information from a software application in one node moves through a network medium to a software application in another node. The model was developed in 1984 by ISO, and is today considered as the primary architectural model for intercomputer communications. The OSI reference model defines a framework for implementing network protocols into seven layers. The seven layers can again be coarsely divided into lower layers (1-4) which focus on data transport, and upper layers (5-7) which focus on the application [2, 12].

7. Application: Specifies the access to lower layer functions and services. Provides functions to the user interface.

6. Presentation: Specifies the representation of data, coding type, compression, and defines used characters.

5. Session: Specifies mechanisms for establishing, managing, and ending connections between two nodes. Dialing control and synchronization of session connection.

4. Transport: Specifies how to ensure reliable data transport. Sequencing of application data, controls start/stop of transmission, sequencing, provides error detection, correction, end-to-end recovery, and clearing. Software flow control between networks.

3. Network: Specifies packet format and routing. Establishes/maintains connections over a network & provides addressing, routing, and delivery of packets to host.

2. Data link: Specifies frame organization and transmittal. Data frame, error detection & correction, sequence control, and flow control.

1. Physical: Specifies the basic network hardware. Definition of the electrical & mechanical functions, and procedural attributes used to access, and send a binary data stream over a physical medium.

2.6 Why use Industrial Ethernet?

Industrial Ethernet is based on the Ethernet technology covered in the IEEE 802.3 specification which is implemented in the ISO/IEC 8802-3 standard. Using Ethernet for industrial communication benefits from the general properties of Ethernet [12, 43, 46]:

- Support for unicast, multicast, and broadcast
- Standardized infrastructure
- Flexible physical layer - can use optical fiber or twisted-pair cables.
- Theoretical possibility of using one network type from enterprise-level to plant-floor
- High bandwidth (10/100/1000 Mbit/s)
- Support for redundant paths between network devices
- Cost savings (equipment is being mass produced)
- A mature technology - thoroughly tested, and a globally approved standard

The last two properties are perhaps the most important within the oil & gas industry. The oil & gas industry is well-known to be conservative and choose reliable, secure, and long time proven technologies. The interoperability and interchangeability qualities are also a great benefit. An Ethernet-based solution using TCP will ensure high reliability, with several error detection mechanisms, but the transmission time will be regarded as less critical. TCP guarantees for the data integrity [43, Ch. 61]. An Ethernet-based solution using UDP is suitable for applications that require fast and efficient transmission, but compromises with fewer error detection mechanisms. The data integrity cannot be guaranteed [43, Ch. 60]. The weakness of Ethernet is the non-deterministic behaviour due to the media access method called the carrier sense, multiple access with collision detection (CSMA/CD). The CSMA/CD challenge excludes hard real-time behaviour for Ethernet control applications, for more details see [46, Ch. 17]. Ethernet is only non-deterministic if collisions can occur. There exists several solutions both in hardware and software to make it deterministic, for hardware a possible method of eliminating collisions is full-duplex switched Ethernet. This setup suppresses the CSMA/CD routine, and data can be exchanged with no chance of collision. The cost is a small latency by introducing switches. Three other techniques for achieving real-time properties are either to modify the Ethernet standard, the TCP/IP communication stack, or the application layer [46, Ch. 17]. The Ethernet standard (ISO/IEC 8802-3) can be modified by use of the standard specifications such as VLAN tags, prioritization of critical messages, rapid spanning tree, and full duplex operations. Performing modification at the Ethernet standard can however cause interoperability problems. Using the original Ethernet & TCP/IP communication stack, and rather perform modifications at the application layer benefits from being able to use cheap conventional of-the-shelf components. Both Modbus TCP and Ethernet/IP are examples on application layer protocols where the modifications are performed at higher layers.

The ISO 13628-6 standard as mentioned above, specifies requirements for the communication protocol between topside and subsea in section *7.4.6 -Communication protocol* [20]. Some of the requirements states that the protocol must be have high reliability & be suitable, it must have sufficient data capacity, withstand normal noise and disturbances, it must support message "time-out", reception of corrupted messages and "time-out" shall result in retransmission of the message, the messages must have cyclic redundancy check (CRC), and it must be able to use it for loading new software to the SEM. This does not explicitly state that Industrial Ethernet shall be used, but industrial Ethernet supports all the requirements listed in the standard. An industrial Ethernet implementation has more or less become the standard for topside communication.

2.7 Standardizing Subsea Solutions

The oil & gas industry has several group organizations working towards finding standardized solutions within subsea. The purpose for this effort is finding common standards that will lead to cost reductions, improved reliability, and generate product solutions that will eventually remove custom made systems. Also, the uncertainty factor for interfaces will be removed. Some of the group organizations are [42]:

- IWIS (Intelligent Well Interface Standard)
- SIIS (Subsea Instrument Interface Standarization)
- SEAFOM (Subsea Fiber Optical Monitoring)
- MDIS (MCS-DCS Interface Standarization)
- SWiG (Subsea Wireless Group)

IWIS, SIIS, and in a sense MDIS as well are relevant for this thesis. IWIS is described in more details in Section 4.2, and SIIS is described below. MDIS is working towards defining and establishing a common interface between the master control systems (MCS) which is delivered by subsea vendors, and the distributed control system (DCS) on the platform. The relationship between MCS and DCS can be seen in Figure 2.2. In 2013, MDIS chose OPC UA as the unified platform and has since continued the work developing a MDIS standard. MDIS has designed software objects describing major pieces of the subsea equipment. By use of OPC UA, software object has been developed and implementation tests has been performed. In 2015 MDIS held an interoperability test, giving the subsea and DCS vendors a possibility of testing their understanding of the MDIS standard on their products [42, 18].

The subsea instrumentation interface standardization (SIIS) is a joint industry project (JIP) where the aim is to standardize the interface between subsea sensors and the subsea control module (SCM) [31]. The SIIS industry group was established in 2003 after the IWIS industry group (described later in Section 4.2). The SIIS industry group developed a specification describing the mechanical & electrical properties, and helped to establish a communication protocol (CiA 443) for subsea sensors & actuators. The whole specification has been submitted to the ISO 13628-6 standard.

An important aspect of SIIS is the definition of three various device levels, or communication methods between the SCM and the subsea instrumentation [30, 13, 38]. The three different device levels are:

- Level 1 - Analogue Devices (4-20 mA): Simple 2-wire loop powered analogue output sensing devices.
- Level 2 - Digital Serial Devices: Relatively complex sensors or actuators that are connected to the control system in a star topology. These devices are configurable and downloadable. The communication network is CAN bus based on a fault tolerant physical layer using CANopen (CiA 301), and the device profile CiA 443.
- Level 3 - Ethernet TCP/IP (industrial protocol) Devices: Intelligent devices where direct communication access is required between the seabed and the application(s) at topside. This device interface may also be used for sensors requiring bulk data transfer. The communication protocol is TCP/IP over Ethernet. An example is a multiphase flow meter where one would like to stream the data topside.

For more information regarding general device requirements, redundant sensors, EMC and testing requirements defined by the SIIS group see [31].

2.8 Event or Time-Triggered Communication

2.8.1 Time-Triggered Communication

Time-triggered communication is performed when data is collected through periodic exchange of messages. This architecture offers predictability, and high dependability (missing messages are detected immediately). Compared to the event-triggered solution it tends to use the most network traffic for exchanging data, and result in the most expensive solution. The reason for the latter is mainly the careful planning during the design phase. For an embedded system an analytical scheduling test is necessary to perform in advance. The analytical test is a systematic & constructive test that is used for constructing appropriate execution schedules. In order to perform the analytical test you need to know all the environmental situations, all task parameters, and all system parameters. However, human errors may occur and situations may be overlooked. These situations will not be handled at all. A disadvantage of the time-triggered communication is that it tends to offer very little flexibility [21, 46, 39].

2.8.2 Event-Triggered Communication

Event-triggered communication is a processing activity that is initiated by the sending device as a consequence of the occurrence of a significant event. The signalling of event can be realized by use of interrupt mechanisms or polling. The event-triggered communication does not require a detailed planning phase with an analytical schedulability test, but a response time analysis must be performed. However, the communication setup must be tested extensively. Especially peak load scenarios where all activities are initiated at exactly the same time. Event-triggered is not so constraint as time-triggered, its in fact quite flexible and easy to modify an operative task or add a new task to an existing node. The event-triggered communication is also well suited for sporadic messages. A possible disadvantage of using event-triggered communication is when a device(s) generates a message storm. This could lead to problems for the communication system if certain protective measures are not taken. [21, 46, 39].

2.8.3 Discussion

To summarize, time-triggered communication is most suited for systems with a need for deterministic behaviour, and periodic update rate. Time-triggered communication is the preferred system architecture for safety-critical systems [46]. Event-triggered communication is suited for transferring event information in systems that needs to offer flexibility and must handle sporadic message sending. The usage of the two obviously depends on the system, and some buses such as the time-triggered CAN (TTCAN) tries to combine the advantage of both concepts [46]. The topside controller in a subsea control system will benefit from a periodical update of data by use of time-triggered communication, while both an event and time-triggered solution could be implemented to handle the sensor data management at the SEM. Since process data related to oil & gas production is sometimes implemented with a safety function, the predictability & dependability of time-triggered communication is highly advantageous.

There exists time-triggered protocols (TTP) designed for embedded real-time control applications. The various protocols are engineered to achieve soft or hard real-time. The great benefit is that they have redundant communication channels, where one of the channels is allowed to fail without affecting the time-triggered communication services. Thus, it offers fault tolerance. More documentation regarding event and time-triggered communication see [21, 46, 39].

Chapter 3

Topside Communication

3.1 Intro

Modbus TCP and Ethernet/IP are two globally accepted industrial Ethernet protocols frequently used within various industries, including the oil & gas industry. OPC UA is added to this section as a suggestion from OneSubsea Processing as a possible way of performing prospective data exchange. As mentioned in Section 2.7 OPC UA is chosen by MDIS as the standard between the MCS and DCS. Using OPC UA longer down the hierarchy should (in theory) provide a solution that are more flexible and easier to integrate.

This chapter starts of with an introduction & evaluation of Modbus TCP, Ethernet/IP, and OPC UA for data exchange on the topside - subsea link. At the end of this section there is a comparison of the industrial protocols based on some of the relevant criterion from OneSubsea Processing.

3.2 Modbus TCP

3.2.1 Introduction

Modbus is an application-layer protocol which means it only defines rules for organizing and interpreting data as a message structure, its independent of the data transmission medium. Since first developed by Modicon in 1979, Modbus has become an industry standard for transfer of discrete I/O information and register data. The data is organized in 16 bit unsigned registers or as single bits. Within the Modbus family there exists three various protocol types: Modbus RTU, Modbus ASCII and Modbus TCP. The Modbus protocol was originally intended for serial transfer but, Modbus TCP was designed to allow industrial equipment to communicate on a Ethernet based network. Modbus TCP embeds a Modbus frame into a TCP frame. This results in a connection-oriented transaction where every request expects a response. The function of the transmission control protocol (TCP) is to make sure all data packets are received correct, while the Internet protocol (IP) performs correct addressing and routing for messages [25].

The Modbus protocol uses the term client (e.g. PLC, host computer) and server (e.g. SEM, valve) between nodes connected to an TCP/IP network. Only the client can initiate transactions. The client sends a request to the server telling it to transfer information or perform a command. The server responds by supplying the requested data, or by execute the command given by the client and then send a echo reply. Due to the transport protocol (TCP) acknowledgment messages are sent back to the client when the message is received by the server. The client - server model is based on four types of messages, as illustrated in Figure 3.1. An explanation of the message types is found in Table 3.1.

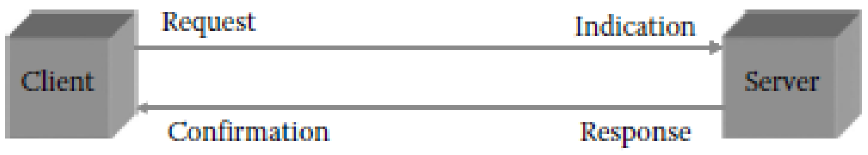


Figure 3.1: The client-server communication model [46, Ch. 10].

Modbus is a request/reply protocol which is considered to be easy to understand, and relatively simple to implement. Its possible to only implement the required message types which will

Modbus Message Type	Description
Request	A message sent onto the network by the client to initiate a transaction
Indication	The request message received on the server side
Response	A response message sent from the server side
Confirmation	The response message received on the client side

Table 3.1: The four basic message types in Modbus

reduce the complexity, and the footprint. This can be crucial for an embedded system where the resources is limited. Modbus is quite scalable and can be implemented at embedded systems and computer systems. The Modbus specification [25] is open and does not require any license fees.

3.2.2 Protocol Description

The basis of the Modbus protocol is the definition of the protocol data unit (PDU). The PDU exists of a function code and a data value. The Modbus function code is a one byte data unit, where value codes are given in the range of 1 to 255. The function code in the message sent from the client to the server informs the server what kind of service to perform. Examples on operations performed by use of the function code are read & write of bits or registers, and diagnostic. The data field of a message sent from a client to a server contains additional information that the server uses to perform the action defined by the function code.

Modbus TCP also encapsulates a Modbus application protocol (MBAP) header which is used to identify the Modbus application data unit (ADU). A graphical representation of the Modbus message structure is illustrated in Figure 3.2. The MBAP header is 7 bytes long and contains the four data fields specified in Table 3.2.

Figure 3.3 illustrates how an ADU packet is passed down the different layers according to the OSI reference model. In order to include the functionality at each layer, the different layers adds its own header to the front of the packet and thereby increasing the overall data size. The ADU is first wrapped into a TCP packet and then a IP packet. The Ethernet functionality is included in layer 1 & 2.

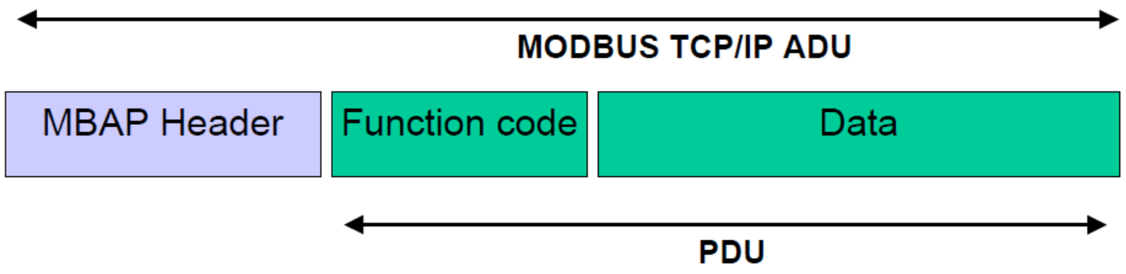


Figure 3.2: Modbus message structure [24].

Length (bytes)	**Data-field**	**Description**
2	Transaction Identifier	Used for transaction pairing when multiple messages are sent along the same TCP connection without waiting for prior response. This is performed by inserting a TCP sequence number in order to create unique identifiers for each TCP request.
2	Protocol Identifier	This field is used for intra-system multiplexing. The ID for Modbus services is 0.
2	Length	This field contains the length of the rest of the packet. This is found by taking the size of the PDU and adding the Unit Identifier byte.
1	Unit Identifier	Used to identify a remote server located on a non TCP/IP network. But, by use of Modbus TCP the server is addressed by use of its IP address. The ID value is set to 0x00 or 0xFF and ignored.

Table 3.2: Description of the MBAP header data fields. Adapted from [24] and [46].

More Details

For an in-depth study of Modbus TCP its recommended to use the *Modbus application protocol* [25], and the *Modbus messaging on TCP/IP implementation guide* [24]. However, for a more practicable approach, see the implementation of the Modbus TCP application layer on embedded microcontrollers in Section 5.1.2.

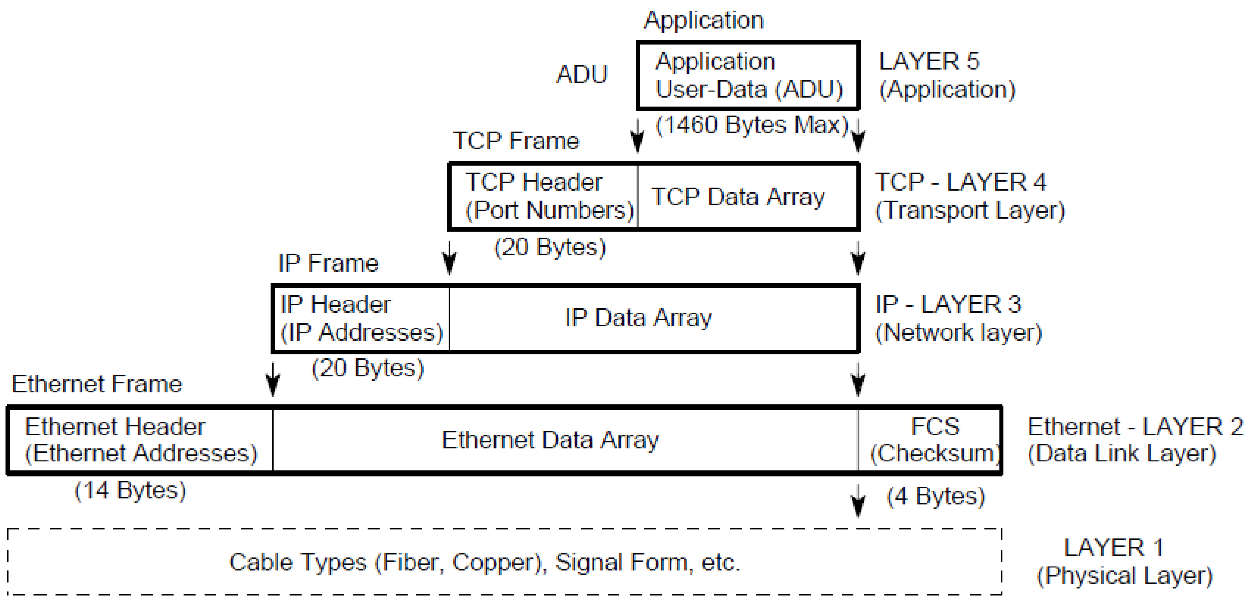

Application
ADU
Application User-Data (ADU)
LAYER 5 (Application)
(1460 Bytes Max)
TCP Frame
TCP Header (Port Numbers)
TCP Data Array
TCP - LAYER 4 (Transport Layer)
(20 Bytes)
IP Frame
IP Header (IP Addresses)
IP Data Array
IP - LAYER 3 (Network layer)
(20 Bytes)
Ethernet Frame
Ethernet Header (Ethernet Addresses)
Ethernet Data Array
FCS (Checksum)
Ethernet - LAYER 2 (Data Link Layer)
(14 Bytes)
(4 Bytes)
Cable Types (Fiber, Copper), Signal Form, etc.
LAYER 1 (Physical Layer)

Figure 3.3: Modbus encapsulation according to the OSI model [2].

3.3 Ethernet/IP

3.3.1 Introduction

Ethernet/IP (*IP* stands for *Industrial Protocol*) is an application-layer protocol which transfer common industrial protocol (CIP) messages inside a UDP/IP packet by use of Ethernet. CIP is a versatile object-oriented protocol used by several industrial protocols such as Ethernet/IP, DeviceNet, ControlNet and CompoNet. CIP defines objects for the messaging application and objects that describes the devices [43]. Figure 3.4 illustrates the organization of the library of the CIP network specification. By using the CIP Sync profile tight time synchronization in all network equipment can be achieved. This tight time synchronization by use of background messages between the clock master and clock slaves provides high accuracy, and enables Ethernet/IP to be used in applications with strict real-time requirements [26]. The CIP enables an upper layer functional security application called CIP Safety (See Figure 3.4). CIP Safety implements integrity mechanisms to achieve fail-safe communication between nodes and safety-components. CIP Safety is certified as a safety application for use in applications in systems needing to meet the requirement of safety integrity level (SIL) 3 in accordance with IEC 61508.

The CIP allow users to integrate messages and service operations on Ethernet supported networks. The CIP defines layers 5-7 according to the OSI reference model described in Section 2.5.

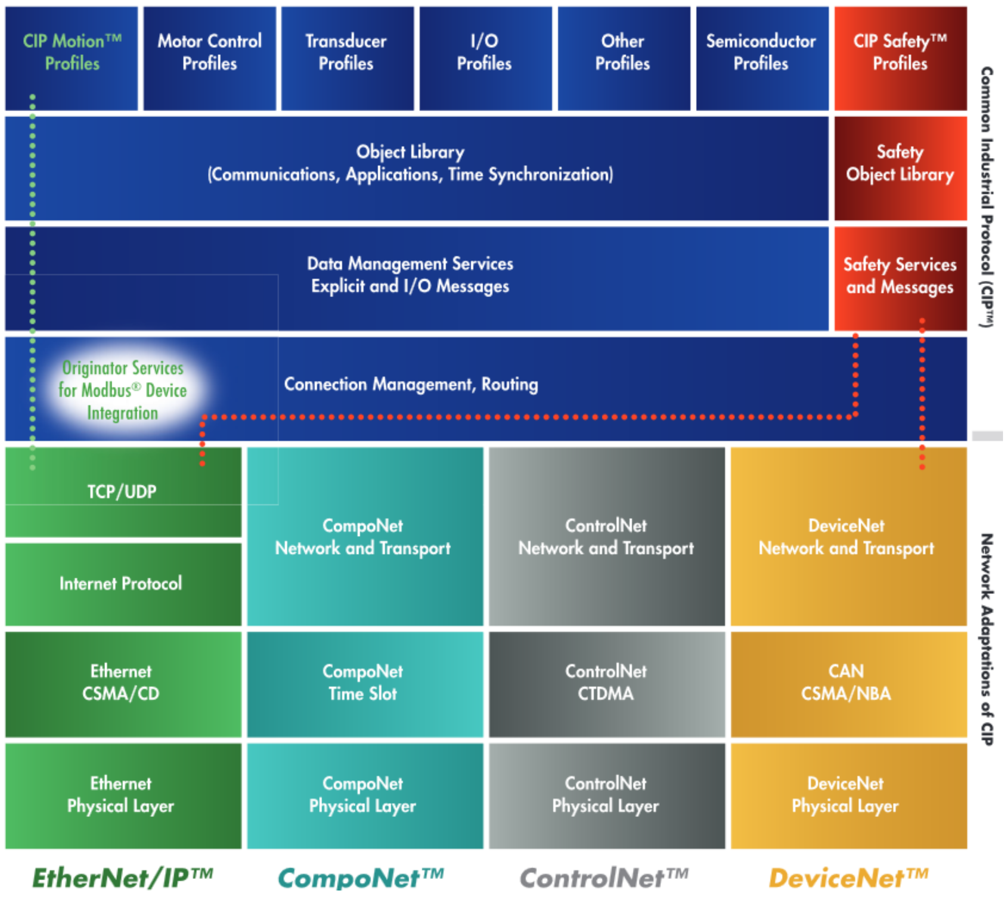

CIP Motion™ Profiles
Motor Control Profiles
Transducer Profiles
I/O Profiles
Other Profiles
Semiconductor Profiles
CIP Safety™ Profiles
Object Library (Communications, Applications, Time Synchronization)
Safety Object Library
Data Management Services Explicit and I/O Messages
Safety Services and Messages
Originator Services for Modbus® Device Integration
Connection Management, Routing
TCP/UDP
Internet Protocol
Ethernet CSMA/CD
Ethernet Physical Layer
CompoNet Network and Transport
CompoNet Time Slot
CompoNet Physical Layer
ControlNet Network and Transport
ControlNet CTDMA
ControlNet Physical Layer
DeviceNet Network and Transport
CAN CSMA/NBA
DeviceNet Physical Layer
EtherNet/IP™
CompoNet™
ControlNet™
DeviceNet™
Common Industrial Protocol (CIP™)
Network Adaptations of CIP

Figure 3.4: Network adaptions of the common industrial protocol (CIP) [27].

The CIP defines a producer - consumer model architecture and not the traditional client - server model as described in Section 3.2.1. The producer - consumer communication model can in several cases make the information exchange more effective. The producer (sending device) broadcast messages on the network for consumption by one or several consumers (receiving devices). Consumers determine it they should receive data in a message based on the identification (ID) field in the packet. Producers can be setup to generate data at pre-established rates, making it unnecessary to issue a request message each time [46].

3.3.2 Protocol Description

As mentioned above, data in Ethernet/IP is represented using objects, and each node in the network is said to be a collection of objects. This abstract object modeling is used to describe communication services, the external visual behaviour of a CIP node, and a common means where information within CIP products is accessed and exchanged [46]. Within CIP there is a large collection of pre-defined object types where each object is assigned an object-ID. The main object types are required objects (e.g, identity, parameter), application-specific (e.g, digital/analog input, motor data), and vendor-specific (e.g, TCP/IP interface, parallel redundancy protocol). Figure 3.5 illustrates the relationship between class, instance, and attributes for a CIP network. Node ID #4 consists of two object classes (#5 and #7), for object class #5 there are two instances (#1 and #2). The data available at instance #2 can be found in attribute #2.

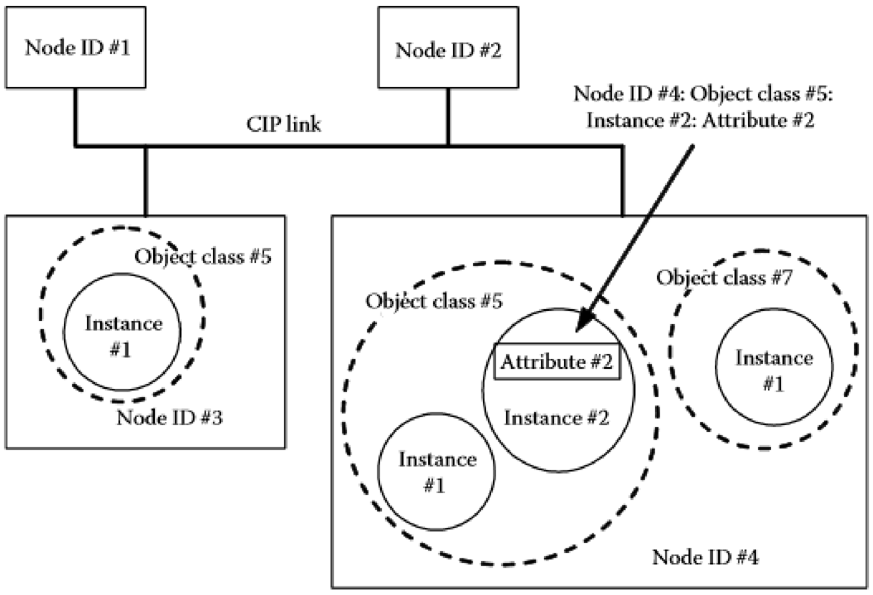


Figure 3.5: Object Addressing in the common industrial protocol (CIP) [46, Ch. 9].

CIP is a connection based protocol coarsely categorized into two types of message types [46]:

- Explicit - Simple point-to-point connections message between nodes, typically used for request/response transactions by use of TCP. The data in the request message explicitly defines what service and objects that are being requested. The data is typically not time-critical information, such as initialization and configuration data.

- Implicit - Special purpose communication between a producing application and one or several consuming applications (producer-consumer transactions) by use of UDP. Consuming nodes determines the usage based on the connection identification. Implicit message is typically real-time specific messages used for exchange of variables such as I/O data and heartbeat signal.

Implicit messaging by use of UDP includes very little overhead and is considered to be more efficient and have better performance. It also uses less resources compared to the explicit message type. Message addressing between nodes are performed by using the components in Table 3.3 which are the same components as in Figure 3.5.

Component	**Description**
Node Address	An Integer ID that is assigned to each node on the network. The node address is the IP address when implemented on Ethernet/IP.
Class Identifier	An Integer ID value assigned to each object class accessible from the network.
Instance Identifier	An Integer ID used to identify an object among all instances of the same class.
Attribute Identifier	An Integer ID assigned to a class or instance attribute
Service Code	An Integer ID which denotes an action request for a particular object instance or object class.

Table 3.3: Components used for message exchange

3.4 OPC UA

3.4.1 Introduction

OPC unified architecture (UA) is the newest industrial communication standard from the OPC Foundation. OPC technology is well known in the industry to provide flow of information among systems from various vendors. The OPC UA application layer is based on a service oriented architecture (SOA) where clients and servers implements sets of services used for handling communication and exchanging process data [17]. OPC UA is specified in 13 parts in the IEC 62541 standard. Part 3 and 4 can be described as the most important as they describe how to model and access information. The UA standard was released in 2008 and replaced the older OPC Classic specification. The background for developing a new standard was mainly the dependency towards the not so flexible Microsoft DCOM/COM platform, but the factors mentioned below also played a significant role [22]:

- Lack of security mechanisms against threats such as viruses
- Not suitable for use on the Internet (firewall challenges)
- Insufficient data models (lacks the ability to represent the data, information and relationship between data items and systems)
- No configurable time-outs

OPC UA and OPC Classic both provides interoperability between systems from several suppliers. OPC UA however offers certain benefits compared to OPC Classic, such as [22]:

- Scalability! The OPC UA specification offers a comprehensive set of capabilities, and servers may implement a subset of these capabilities in order to cover its need. This makes it possible to implement OPC UA at devices from embedded level to enterprise level.
- Platform independent
- A flexible information model architecture, which supports complex modelling of information in the UA address space.

- Internet/Firewall friendly
- Reliability (robust, fault tolerant and redundant)
- Secure communication with access control
- All previous specifications (data access, alarms & events, historical data access) put together into one
- OPC UA is backward compatible which means existing data exposed by OPC COM servers can easily be mapped and exposed via OPC UA.

OPC UA sends messages in a client - server communication model as illustrated in Figure 3.6. The system architecture models clients and servers as interacting partners. An application may combine servers and clients components to allow interaction with other servers and clients. An example of this can be seen at the intermediate level between the enterprise and operations network, or between the operations and the plant floor networks in Figure 3.6. An intermediate UA client extract and process data from lower level UA servers, this data can then be re-modelled and integrated into the intermediate UA server's address space [43, Ch. 57]. OPC is an example of a protocol that offers the technology for crossing the line between the plant floor network and the enterprise network.

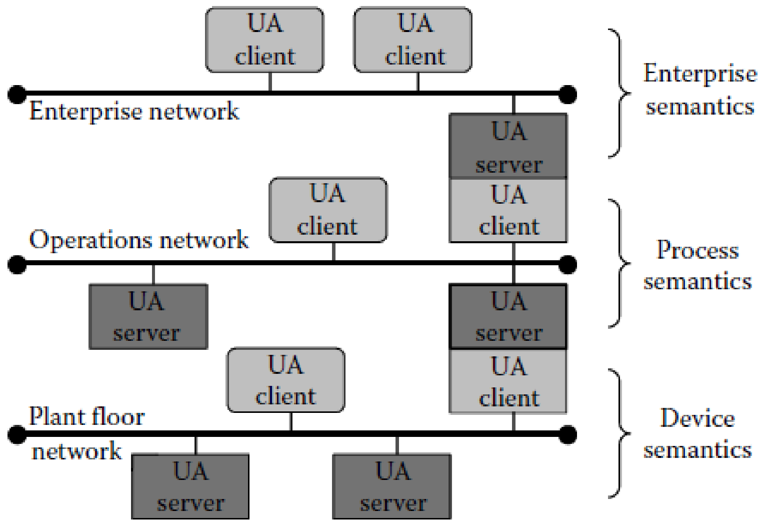


Figure 3.6: OPC UA network of clients and servers [43, Ch. 57]

The OPC UA specification is layered to isolated the core design from the underlying computing technology and network transport. It does not define how servers should acquire data and how clients may use the received data [17]. By only defining the upper layers of the OSI model, it will support mapping to future technologies without modifications to the basic design. Part 4 of the UA specification describing services represent OSI layer 7 - application. Part 6 which describes transport mapping of UA services to network protocols and data encoding match OSI layer 5 and 6 information. Data encoding is performed by UA binary structures or XML text documents. For transport mapping - UA provides two options, either TCP or SOAP Web services or HTTP [19, Part 1]. An analysis of the OPC UA performance is executed in the article [11]. This research article presents performance evaluation on various scenarios, where the respective scenarios are related to different levels of security and bandwidths. The test results shows that transport mapping using TCP performs better than SOAP at all of the tested security levels.

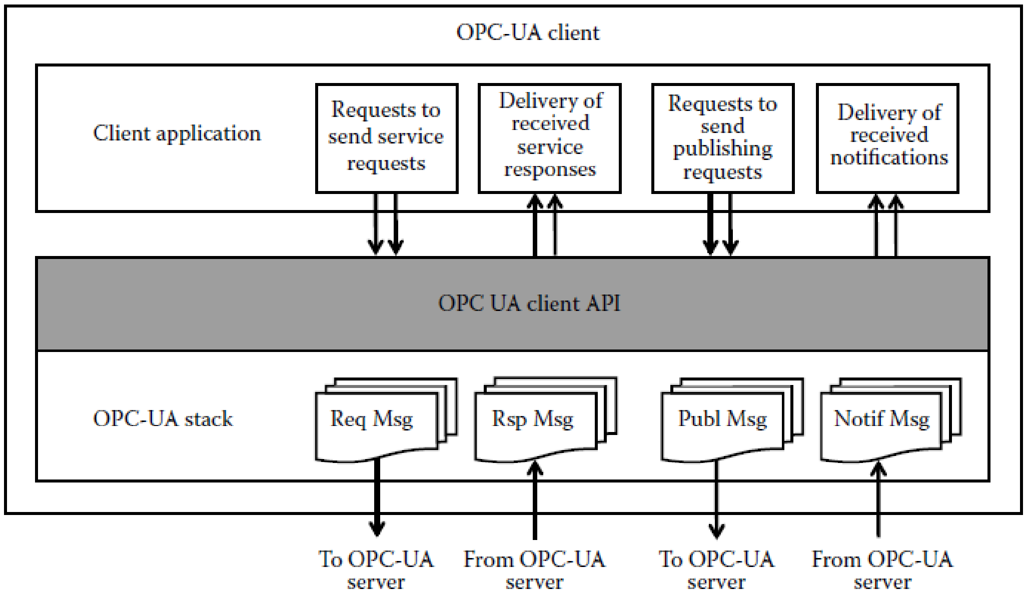


Figure 3.7: OPC UA client architecture [43, Ch. 57]

3.4.2 UA Client Architecture

The client application in Figure 3.7 is the function that represents an UA client. The UA Client is depending on the OPC UA client application programming interface (API) which provides service mechanisms according to the UA specification to use for send and receive of request/response messages. The article [17] discusses the performance effects on the CPU load & server/-client update times due to increasing number of UA clients.

3.4.3 UA Server Architecture

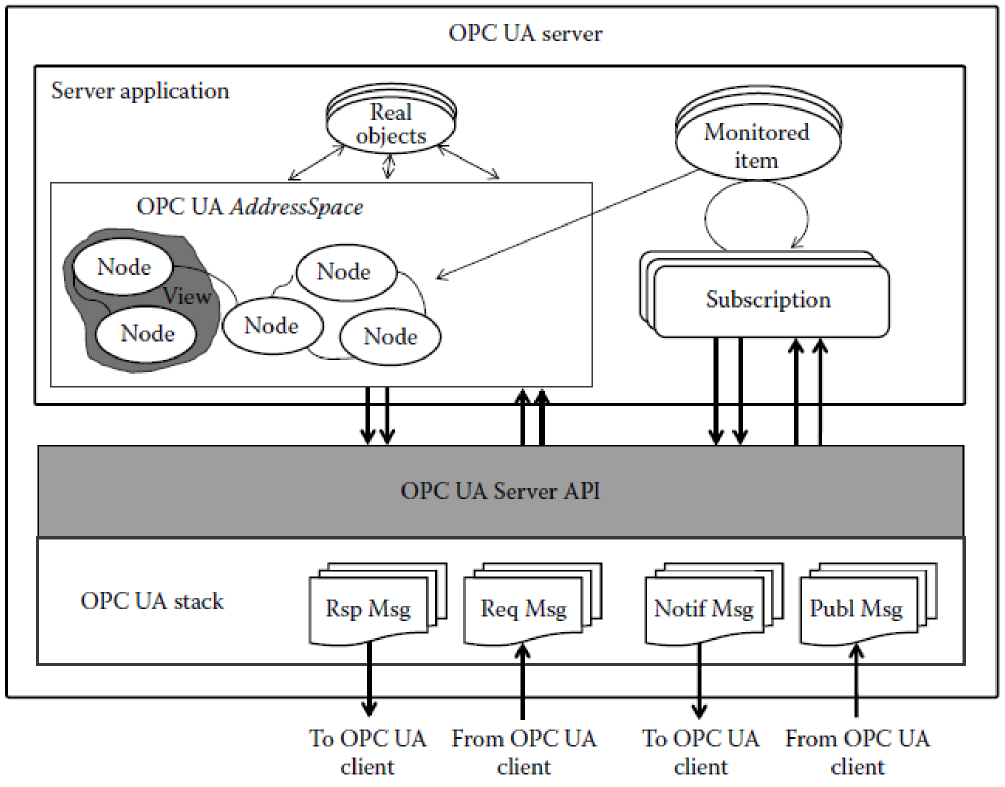


Figure 3.8: OPC UA server architecture [43, Ch. 57]

An OPC UA server is used to model data, information, processes, and systems as objects. By adding objects to an address space, the content can be represented to a client as a set of nodes connected by references [28]. The central elements of the OPC UA server can be seen in Figure 3.8. The server application implements the necessary functionality to act as an OPC UA server. The OPC UA server uses the API to send and receive OPC UA messages from/to the OPC UA client application [28].

The concept and usage of the address space is essential in OPC UA. An object model defining objects in terms of variables and methods, as illustrated in Figure 3.9 is used for documenting the elements in the address space. The Nodes in the address space are instances of objects as de-

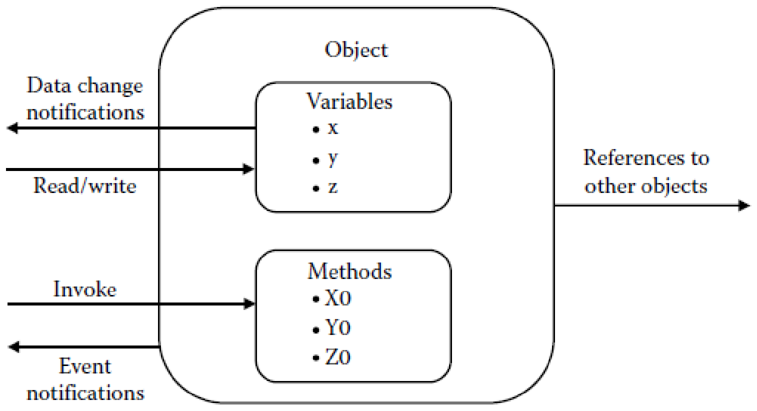


Figure 3.9: OPC UA object model [43, Ch. 57]

fined in the object model. As illustrated in Figure 3.8, the Nodes in the address space represents real objects (physical or software objects). Such a node consists of attributes and references as shown in Figure 3.10. Attributes are available for specific node classes and are used to describe node-IDs, values, data types, display/browse names, etc. The references is used to relate Nodes to each other, they decide the visibility for the UA client. Access to the Nodes is performed using the OPC UA services.

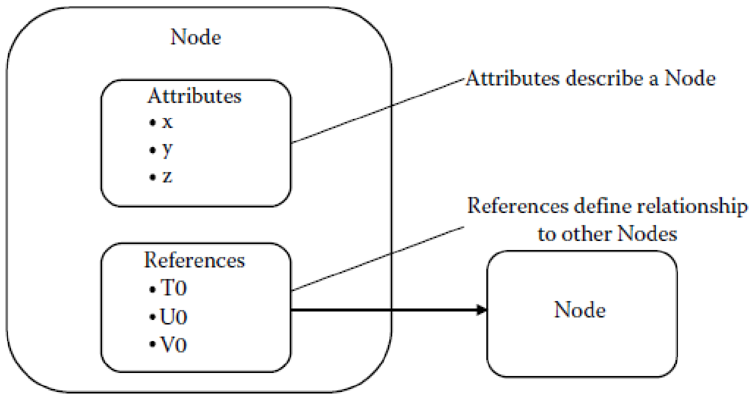

Node
Attributes
• x
• y
• z
References
• T0
• U0
• V0
Attributes describe a Node
References define relationship to other Nodes
Node

Figure 3.10: OPC UA AddressSpace node model [43, Ch. 57]

The object has definitions (e.g. physical units) and references to each other. Part 3 of the UA specification describes the OPC UA address space [28]. An OPC UA client can access the Nodes in the address space using UA services (interfaces and methods) as described in Part 4 of the UA specification [43, Ch. 57]. Among the diversity of helpful services in Part 4, several communication exchange patterns are specified. This includes request/response, Subscription, and server-server services.

OPC UA offers an elegant functionality called Subscription which can be enabled using the Publisher services. An OPC UA client can use the Subscription concept to establish a periodical function that will inform about specific attributes of OPC UA tags provided by the UA server. The client can modify the periodical publish interval in order to decide when it wants to receive repetitive information from the UA server regarding the OPC UA tags. This way of exchanging information is very predictable as the information will arrive periodically, and a transmission error ending in a lost message will be easy to detected. The timestamp and quality markers will be update when messages are received.

The Monitored items in Figure 3.8 is a queue where items of interest are added. These items are given a user-defined sampling rate for when the real items are checked for alarms, events, data changes, aggregates, etc. In case of an occurrence, a notification is generated and put into the Subscription queue. The Subscription will transfer a message regarding this item when the publishing interval elapses. A variety of choices for the sample and publishing interval can be configured, more information can be found in Part 4 of the UA specification [28]. The implementation of the Subscription functionality can reduce the amount of transferred data immensely, especially if the data changes infrequently. When the publish interval for the UA tag elapses and there is nothing to report, a KeepAlive notification will be sent to the UA client, to indicate that no data has changed and the server is still running. The server can then hold the same value, but it will update the timestamp for the specific OPC UA tag [43, Ch. 57]. The article [17] discusses the performance impact for increasing number of Monitored items.

For an in-depth study of OPC UA, its recommended to use the OPC UA book [22] or the specification from the OPC Foundation Web site [28]. The OPC Foundation web site also offers a lot of information regarding implementation and product availability.

3.5 Evaluation - Comparison and Discussion

Another relevant protocol for communication between topside & subsea is the FMC 722 subsea automation protocol. This is a proprietary protocol developed by FMC Technologies which is the leading provider of subsea systems. FMC Technologies typically deliver systems (e.g subsea trees) that are more integrated into the platform DCS and has more strict requirements for safety critical functions. Due to difficulties of finding, and being allowed to publish technical information of this proprietary protocol, further details are omitted in this thesis. This section starts of with discussions related to central evaluation points for performance & features, before ending with a conclusion of a recommended topside communication protocol. OPC UA is not relevant for all of the topics, and is therefore excluded in some of the discussion topics.

3.5.1 Theoretical Performance

All industrial communication networks are described in the IEC 61158, and the companion IEC 61784 describes how to build a specific communication network using the IEC 61158 [27]. The IEC 61784-2 profiles for industrial communication network is based on the ISO/IEC 8802-3 (Ethernet). The Ethernet standard can provide functionality such as VLAN tags, prioritization for critical messages, rapid spanning tree (relevant for fault tolerant architecture), and full duplex operations (collision free message exchange) to ensure real-time properties for Ethernet based networks. These functions eliminates delays due to the CSMA/CD function as described in Section 2.6. Several of the profiles in IEC 61784 are given performance indicators. The performance indicators are meant to specify the capabilities of a RTE communication network and RTE end devices [19]. Some of the indicators are delivery time, number of switches or end stations, and throughput RTE. Rest of the performance indicators can be seen in Table 3.4. The performance indicators provides the possibility for users of the RTE network to set requirements for their different applications. Complementary information on performance indicators can be found in [46, Ch. 17].

The Modbus TCP as default using the server-client communication model has poor real-time response. However, the IEC 61784 standard which specifies profiles for industrial communication networks describes a real-time extension for Modbus by enabling the Real-Time Publisher Subscriber (RTPS) protocol. More details on RTPS can be found in IEC 61784-2 [19, Comm. Profile 15/2], a complementary description can also be found in [46, Ch. 17]. The article [16, p. 1122] however states that *the RTPS protocol is not used very much in practical industrial applications today, and therefore it is not known exactly what sort of performance this protocol really has to offer*. Modbus TCP is one of the profiles in the IEC 61784 where the performance indicators are not quantified. The reason for this is given in the *Modbus messaging on TCP/IP implementation guide* [24, Ch. 4.4.1.4] which states that *There is deliberately NO specification of required response time for a transaction over Modbus TCP. This is because Modbus TCP is expected to be used in the widest possible variety of communication situations, from I/O scanners expecting sub-milliseconds timing to long distance radio links with delays of several seconds*. Hence, Modbus is too dependent on the network type, topology, hardware and the actual exchange management in the application layer implementation in order to quantify performance indicators [35, 19, 36].

Performance indicator	Profile 2/2	Profile 2/2.1
Delivery time	130 μs - 20.4 ms	130 - 190 μs
Number of end-stations	2 - 1024	2 - 90
Number of switches between end stations	1 - 1024	1 - 4
Throughput RTE	0 - 3.44 M octets/s	0 - 3.44 M octets/s
Non-RTE bandwidth	0 - 100 %	0 - 100 %
Time synchronization accuracy	–	≤ 1 μs
Non-time-based synchronization accuracy	–	–
Redundancy recovery time	–	–

Table 3.4: Performance indicators for default Ethernet/IP (Profile 2/2) and Ethernet/IP with CIP-Sync (Profile 2/2.1) as specified in IEC 61784-2.

The Ethernet/IP protocol as described in IEC 61784, has the capability of achieving good RT properties. Of course, also Ethernet/IP is dependent on the topology, hardware, and efficiency of implementation. However, Ethernet/IP has quantified performance indicators in the IEC 61784. The performance indicators for default Ethernet/IP can be found in Table 3.4 in the column for Profile 2/2. By enabling the CIP Sync profile specified in IEC 61784-3 (Profile 2/2.1) its possible to achieve even stricter time constraints. CIP Sync enables clock synchronization, and as illustrated in Table 3.4 it can achieve a delivery time in the range of 130 - 190 μs [19, Comm. Profile 2/2.1]. This makes it highly desirable for drives, motion and robotics usage.

The OPC UA specification does not specify communication parameters such as performance indicators. As mentioned in Section 3.4, OPC UA is a higher layer protocol where the OPC UA communication stack is not linked to any specific technology. OPC UA consists of a diversity of services and mechanisms which can be enabled/disabled based on the available resources [10]. All of these services also makes it difficult to determine any form of normative performance indicators for OPC UA.

3.5.2 Studies of the Performance

As mentioned in the section above, the Modbus TCP protocol deliberately does not have any specification of required performance for a transaction. The reason for this is the diversity of the protocol. Modbus TCP is expected to be used in a vast numbers of communication situations from embedded systems with milliseconds demands, to radio links with response within seconds.

A theoretical study of the Ethernet/IP and Modbus TCP performance in multicast (one-to-many) communication is performed in the research article [36]. The article derives simple communication scenarios where expressions for the theoretical minimum cycle time are found for the major industrial Ethernet protocols available on the market (Ethercat, Profinet, Ethernet/IP and Modbus TCP). The communication scenarios is setup to consist of one controller (client) that exchanges data with several nodes (servers). By illustrating a space-time diagram a relationship between the transmission delay, the network device latency, the propagation delay, the link capacity, the payload, and number of servers are found. The combination of these parameters expresses the minimum cycle time, which is the communication time required by the controller to collect and update the data memories of all sensors and actuators. The main interest in the article in regards to this thesis are the comparison of respectively Modbus TCP and Ethernet/IP. The expression for theoretical minimum cycle time for Modbus TCP is given in Equation 3.1a, and the expression for minimum cycle time for Ethernet/IP is found in Equation 3.1b. Both

Equations are given in [36, Section 2.4-2.5].

$$\Gamma_{Modbus_{TCP}} = n\left(8\frac{181+x}{C} + 2(2\delta + l)\right) \tag{3.1a}$$

$$\Gamma_{Ethernet/IP} = 2\delta + l + 8n\frac{84+x}{C} \tag{3.1b}$$

Where n is the number of network devices (servers), x is the payload (bytes), C is the link capacity (bits/sec), δ is the propogation delay (sec), l is the network device latency (sec). The Ethernet capacity/bandwidth is typical C = 10 or 100 Mbit/sec, and a typical packet consists of x = 100 bytes (payload). The packet size usually varies, but for comparison the size is regarded as constant. The network device latency represents a delay due to components in the network, e.g. a switch, router or hub. This value is typically found in data sheets. The article [36] uses the values: $l_{Modbus\ TCP}$ = 1μs, and $l_{Ethernet/IP}$= 3μs at an link capacity of 100 Mbit/sec. The minimum cycle time for Modbus TCP and Ethernet/IP expressed in Equations 3.1a and 3.1b are plotted in Figure 3.11. From the figure we see that Modbus TCP is the slowest Industrial Ethernet protocol of the two, at both 10 Mbit/sec and 100 Mbit/sec.

The article [36] concludes that Modbus TCP is not suited for multicast communication (one-to-many) due to the server-client communication model and use of TCP as transport protocol. Sending response/request messages with TCP generates a lot of "unnecessary" acknowledgement response. Ethernet/IP however supports the producer/consumer communication model by use of UDP as transport protocol. By using UDP, Ethernet/IP achieves fast & efficient data transfer, but it lacks some error detection mechanisms. Modbus using TCP guarantees the data integrity, but generates a lot of message overhead and acknowledgement messages.

In general OPC UA lacks literature documenting the performance specifications. As mentioned in the previous section the UA specification does not state any performance parameters because of the diversity of implementations, and due to the fact that its technology independent [17]. Due to the large selection of services and mechanisms provides by the UA specification it can be implemented on embedded systems and enterprise solutions. Some studies have been performed on how these services and mechanisms impacts the parameters of performance.

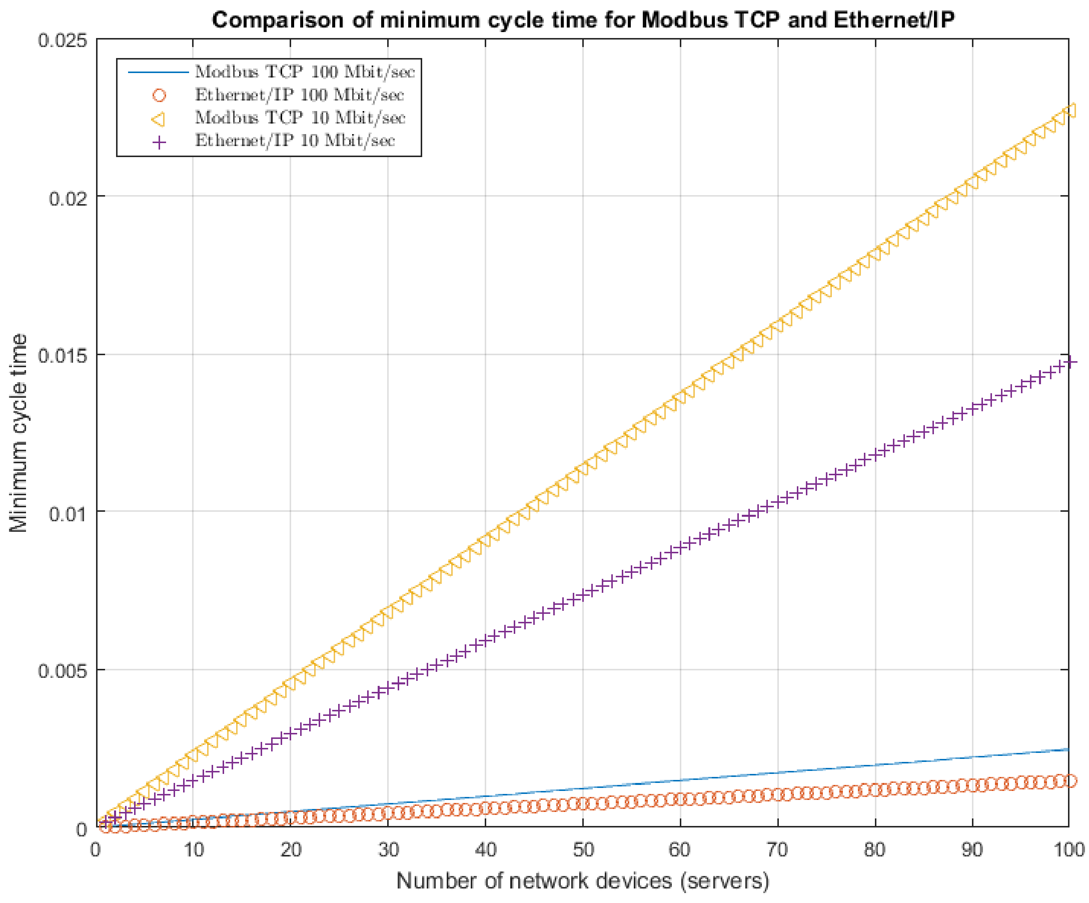


Figure 3.11: Comparison of minimum cycle time for Modbus TCP and Ethernet/IP

Examples are:

- Performance impact on the transport protocols (TCP and SOAP) at different security levels are discussed in article [11].
- Performance impact on the CPU load, and the server/client update time due to increasing number of clients [17]. Also, the amount of network traffic due to increasing number of Monitored items & clients are considered in article [17].
- Performance impact on the bandwidth and delays due to variations in the publish interval at the Subscription mechanism is considered in article [10].

The article [17] contains an interesting conclusion regarding the use of a Java based OPC UA SDK on a standard 100 Mbit/sec Ethernet network. The network load only reached approximately 2 Mbit/sec which is a bandwidth utilization of only 2.5%. The article assumes that the bottleneck is not the network, but the internal architecture of the OPC UA SDK server and the utilization of hardware resources [17].

3.5.3 Safety Protocol

The requirement for whether or not to implement a safety protocol is decided based on laws & regulations. For petroleum activities at the Norwegian continental shelf, this is based on regulations set by the *The management regulations* [33] (NO: Styringsforskriften) published by the Petroleum Safety Authority Norway. A safety protocol will operate as a barrier and *reduce the possibility of failures, hazard and accident situations occurring and developing.* [33, §5 Barriers]. A safety protocol consists of defend measures to protect against predefined communication errors such as corruption, unintended repetition, incorrect sequence, loss of data, unacceptable delay, insertion, masquerade, and addressing. The safety protocol is realised by adding an additional layer (safety layer) on top of the application layer. A safety protocol does not assume anything about the integrity of the existing network connection, it consider it a *black channel*. The *black channel* is independent of the transmission system characteristics such as communication stack and network devices [43, Ch. 46].

IEC 61784-3 [19] specifies various commercial industrial safety protocols, and all the defend measures like sequence numbering, time stamp, time expectation, connection authentication, feedback message, data integrity assurance, redundancy with cross checking, and different data integrity assurance systems. The combination of these defend measures achieves a safety protocol which is certified for a given safety integrity level (SIL). Insertion of safety related data into the data frames will cause a reduction of the payload.

A safety protocol can in theory be implemented onto all communication protocols, since its an additional layer on top of the application layer. Ethernet/IP however supports the predefined CIP Safety profile which fulfills the requirements for SIL 3. The CIP Safety profile will therefore most likely result in a easier implementation compared to a safety protocol solutions for Modbus TCP. Modbus TCP does not have any predefined safety protocols.

3.5.4 Diagnostics

Another discussion topic is related to diagnostic functions. Is there any predefined specifications supporting diagnostic data for troubleshooting such as link status, utilisation of capacity, collision data, message count, communication error, etc. ? This is typically regarded as important status information to the developers for checking that data packets are arriving correctly.

The *Modbus application protocol* [25] specifies function code 0x08 with several sub-codes as diagnostic information. It can be used for checking the communication system and various internal error conditions within a server. However, the diagnostic function code is only applicable for serial line devices. Hence, Modbus TCP does not have any predefined diagnostic information. Ethernet/IP offers extensive diagnostic information in the Ethernet/IP specification (volume 2, chapter 9) [27]. It also has predefined specifications regarding device type, vendor code, revision number, status and state information.

OPC UA is probably the protocol supporting most diagnostic information. The diagnostic information is split into information per server, per session, and per subscription [22, Ch. 4.4]. Appendix A in the OPC UA specification [28] - part 5 explains the design decisions regarding the modeling of diagnostics information.

3.5.5 Redundancy

The oil & gas industry has strict requirements for availability and reliability. Redundant hardware is a popular method to achieve these qualities. The topside controller and SEM are some example on the redundant hardware in a subsea project. Also, the communication link between topside and subsea is normally based on fiber optics but in some projects, a redundant copper link may also be implemented. As long Modbus TCP, Ethernet/IP and OPC UA are all based on standard Ethernet, they all support redundant paths. The Ethernet standard implements the (rapid) spanning tree protocol, this protocol uses some seconds to perform a network reconfiguration to another active alternative path [43].

OPC UA supports redundancy based on the existence of duplicate client or server applications. These client or servers can be achieved by using data structures and services (modes, roles at initial connection, failover criteria) specified in the UA specification [28, 22]. The redundant server or clients can also be used for load balancing. The specification for Modbus TCP and Ethernet/IP does not include any redundancy mechanisms. The redundant controllers must use vendor specific mechanisms to perform this operation for Ethernet/IP and Modbus TCP.

3.5.6 Availability

Ethernet/IP was originally developed by Rockwell, but is controlled by the ODVA organization today. Regardless, its still backed by Rockwell/Allen-Bradley and has a strong political advantage [23]. Vendors must become members of ODVA, and pay an annular membership fee for the CIP & Ethernet/IP specifications. The Modbus TCP protocol however is quite independent of market forces, and is open & freely available on the Modbus Organization website [25, 24]. Access to the OPC UA specification is given to members of the OPC Foundation. Similarly as for ODVA, an annular fee must be paid to the OPC Foundation. A membership in the OPC Foundation also includes access to the official OPC UA communication stack, compliance test tools, software development kits (SDK), and other helpful tools [1].

3.5.7 Conclusion

The performance of topside communication will in general depend on main factors such as the network topology, hardware, and efficiency of the application layer implementation. Research articles reveals that the best OPC UA performance will be achieved by using TCP as transport protocol. The performance will considerably depend on the number of clients, efficiency of the SDK, number of Monitored items, and the enabled number of service mechanisms. As mentioned above, Ethernet/IP quantifies performance indicators that describes the capabilities of a RTE communication network and RTE devices. The performance indicators are only normative, and is supposed to indicate the achievable performance in a Ethernet/IP network. Modbus TCP

[1] https://opcfoundation.org/membership/benefits/

does deliberately not specify performance indicators due to its wide variety of implementation situations.

By summarizing the introduction and discussion topics for topside communication protocols, its clear that Ethernet/IP has better real-time properties compared to Modbus TCP. Modbus TCP uses a client-server communication model based on request/response messages, and according to Article [14] this is not a suited real-time communication model. The same article recommends the publish-subscribe or the producer-consumer model as communication models for real-time communication. Ethernet/IP supports the producer-consumer model, and OPC UA supports publish-subscribe.

Figure 3.11 illustrates a noticeable difference in performance between Modbus TCP and Ethernet/IP for multicast (one-to-many) communication. At 10 Mbit/sec Ethernet/IP is superior compared to Modbus TCP, as illustrated in Figure 3.11. At 100 Mbit/sec the baud-rate and margins is ten times greater for the transport protocol to handle most of the acknowledgement response, and the difference in cycle time is not so significant. A trade off between the fast & efficient data exchange of an UDP based communication protocol, and the guaranteed data integrity properties of TCP would have yield the best overall result.

All of the protocols can address the same number of nodes since this is determined by the configured IP-subnet. Ethernet/IP benefits from being able to offer more features such as data synchronization in the CIP Sync profile, and the safety protocol - CIP Safety profile. Meaning the Ethernet/IP protocol can be used for implementation of SIL 3 safety applications and time critical applications. While Modbus TCP really is just an implementation of serial Modbus on an Ethernet network. Seen from the developer side, Modbus TCP is thought to be simple to integrate and easy to use for data transfer.

OPC UA was added to this thesis with the purpose of highlighting some of its features for prospective implementation. OPC is well known in the industry for being an interface offering interoperability by allowing process data sharing between systems from different vendors [22]. This is still an important requirement for OPC UA. However, OPC UA offers much more than the previous OPC Classic standard. UA provides an information model architecture which can be used to model complex information [28]. This flexible model architecture is of great interest for modeling detailed information such as complex data types and diagnostic information for subsea components. As mentioned in Section 2.7 MDIS has chosen OPC UA as the unified architecture between DCS & MCS and have already designed software objects describing major pieces of subsea components. Perhaps this is a step towards a complete vertical communication network starting at the subsea instrumentation to the DCS by using only one communication protocol? This concept of converging all network levels is illustrated in Figure 3.6. OPC UA is however not a real-time protocol, and there are hardly any safety critical systems based on OPC UA. Situations where data is not be delivered within a bounded time may occur. This situation is handled by providing timestamps and quality markers which may be used to identify the data that is not up-to-date [17]. An adoption of OPC UA is still predicted to continue within the oil & gas industry [42]. Otherwise, OPC UA provides an elegant functionality called the Subscription concept which is used by the UA client to subscribe for relevant information from the UA server. As mentioned above, this is a recommended real-time communication model. OPC UA also has extensive support for IT security and is engineered to be Internet and firewall friendly. Ethernet/IP and Modbus TCP does not support any higher layer security mechanisms. More information about security measures, see part 2 of the OPC UA specification [28].

Chapter 4

Subsea Sensors & Intelligent Well Devices

4.1 CAN bus (SIIS Level II)

Controller Area Network (CAN) is a bus system initially developed to replace point-to-point cables in the automotive industry. The reason for introducing CAN were the increasing costs, complexity, and weight of the electrical and electronic systems. From a communication point of view - serial CAN bus is a half duplex, message-oriented transmission protocol. It has a producer-consumer communication model where each message is assigned a priority. When two or several nodes starts transmitting at the same time, the message with the highest priority wins the arbitration and can continue the transmission. The other node(s) backs off until the transmitting node is finished. This is called non-destructive bit-wise arbitration and gives CAN real-time properties. It also avoids message collisions. The CAN bus is regarded as having a high level of reliability & robustness. The two main reasons for this are the setup consisting of few components, and the fact that data at the physical layer is represented by differential voltage between two bus lines. The differential voltage results in good noise immunity. CAN has also support for higher layer protocols describing specific devices (meters, encoders, HVAC), it can be setup to be fault tolerant (maintains the functionality although there is one error on the bus) and many error detection mechanisms (CRC, frame check, ACK errors, transmission monitoring). Related to the OSI model, the CAN bus is defined at layer 1 and 2. The protocol is specified in the ISO 11898 standard. Where part 1 describes the data link layer, and part 2 & 3 of the standard specifies the high-speed & the fault tolerant (low-speed) physical layers [8, 43, 15].

Eventually when a certain number of nodes, messages, and network variables are involved, a higher-layer protocol needs to be used in order to structure the data, and provide extended functionality. For instance, handling transmission of data blocks larger then 8 bytes (the data link layer specifies a maximum payload of 8 bytes), and coordinate initialisation of nodes. The system designer needs to specify which identifiers are used for various purposes, and the contents (data types, byte order) to complete the higher-layer protocol. A mature and flexible protocol for this purpose is the CANopen (CiA 301) specification owned by the CAN in Automation (CiA) organization. CANopen describes the exchange of data in a CANopen network with regards to communication services, network management services, the functionality of communicating with specialized devices (device profile), and the application parameters (e.g. process, configuration and diagnostic data). CANopen defines how the CAN message frames consisting of a 11 bit identifier and 8 byte data are interpreted [8, 41, 15] .

CANopen provides a large number of services, these can be divided into four application layer entities [15, Ch.3]:

- CAN-based Message Specification (CMS) - An object-oriented environment for designing user applications.
- Network management (NMT) - specifies a NMT master which is used for initialising, error handling, and supervising of other NMT slaves. If nodes goes offline, sets off an alarm, or sends an emergency message, the NMT must perform a recovery or a shutdown procedure. The NMT uses a heartbeat message to supervise the operation of the nodes.
- Distributor (DBT) - Handles the allocation of message identifiers (COBs). As mentioned above, the identifier determines the priority in a message. The DBT master can be used to change the identifier of the other DBT slaves.
- Layer Management (LMT) - A LMT master can be used to change the settings of certain layer parameters (address, bit-timing, baud-rate) in other LMT slaves.

CANopen implements an object dictionary (OD) which is a table containing configuration and process data with a 16-bit index and a 8 bit-subindex. By using the index and subindex in de-

Index Range	**Description**
0000_h	Reserved
0001_h-$0FFF_h$	Data types
1000_h-$1FFF_h$	Communication entries
2000_h-$5FFF_h$	Manufacturer specific
6000_h-$9FFF_h$	Device profile parameters (E.g. CiA 443)
$A000_h$-$FFFF_h$	Reserved

Table 4.1: Index range for the object dictionary (OD), adapted from [8, Table 41].

fined CAN messages, its possible to read and write information from or to the OD. The OD for instance can be used for configuration of communication parameters, setting correct data types for variable exchange, heartbeat, identification of nodes (vendor ID, product code, serial no., rev. no.), errors and more. Table 4.1 illustrates the 16-bit index as defined in the CiA 301. The definition of data types (INT, UINT, REAL, BOOL) used in variable exchange are set in the index range of 0001_h-$0FFF_h$, while read and write of variables are typically performed through the communication entry [8, 41, 15].

Access of the available data in the object dictionary is performed by using either service data objects (SDO) or process data objects (PDO). The SDO is based on request and response messages by use of a peer-to-peer communication. SDO messages generates a lot of protocol overhead because it sends messages in a segmented structure. The SDO messages carries high volume data with low priority, such as configuration data to nodes. The PDO message type is used to transfer real-time data onto the network, and is a more effective message type compared to SDO. Data is transferred with low volume but with high priority. Several dictionary entries (process values) can be mapped into one PDO. The PDO message type operates in a producer-consumer model, therefore it can send the same data to several nodes in one transmission. Also, the PDO does not generate protocol overhead with its maximum 8-bytes length. The structure of a CANopen device can be seen in Figure 4.1. The application uses the data in the object dictionary to perform its function.

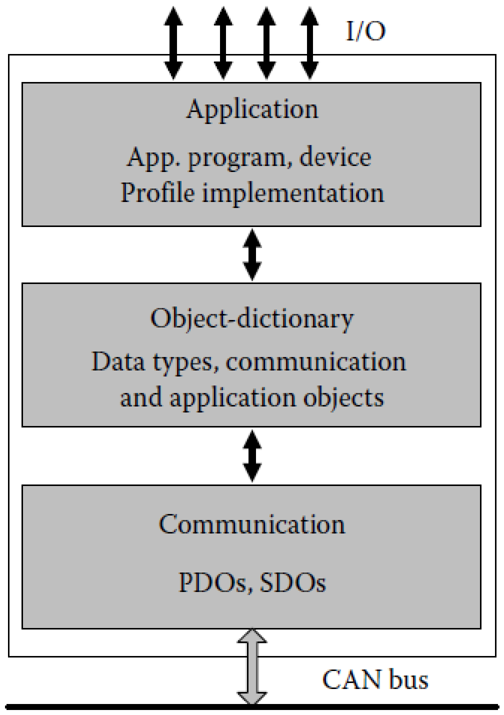


Figure 4.1: The structure of a CANopen device [43, Figure 31.2].

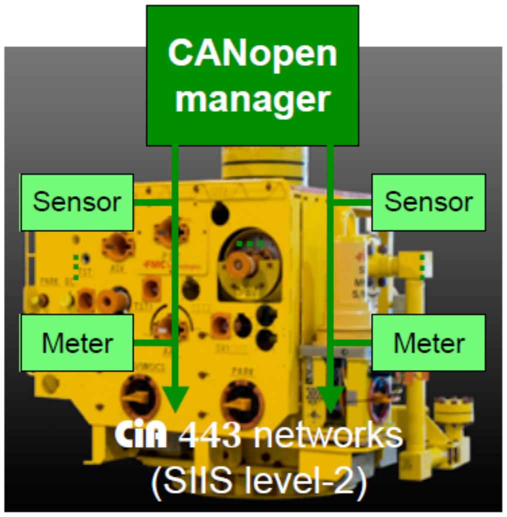


Figure 4.2: Device profile for subsea measurement systems. Figure from [45].

The CiA 301 - CANopen acts as the base of the network by defining an application layer [15], and offers a framework for developing device profiles that provides interoperability between devices from various manufacturers. Specifically for subsea the *CiA 443 - CANopen profile for SIIS level-2 devices* is implemented on top of the CiA 301 - CANopen, as illustrated in Table 4.2. A simplified illustration of the CANopen manager controlling the subsea sensor & actuator network is shown in Figure 4.2. The CiA 443 device profile was developed by the SIIS group (mentioned in Section 2.7) together with the CiA.

OSI-model layer	Description
7 - Application	Device profile (CiA 443) CANopen (CiA 301)
6 - Presentation	–
5 - Session	–
4 - Transport	–
3 - Network	–
2 - Data link	ISO 11898-1
1 - Physical	ISO 11898-3: Fault Tolerant, ISO 11898-2: High speed

Table 4.2: CAN bus used for subsea instruments in reference to the OSI model.

The CiA 443 is a device profile which specifies process data types for measured values, commands, configuration parameters, and diagnostic information. This is data types used for sensor & actuators in a subsea measurement systems such as temperature, pressure, multiphase meters, corrosion/erosion, sand, vibration and pig detectors, injection & control valves, etc. The CiA 443 device profile assumes that devices are implemented with a fault tolerant (ISO 11898-3) physical layer which supports the default bit-rate of 50 kbit/sec, and also 125 kbit/sec. Optionally, it also supports the high speed ISO 11898-2 standard. For more information regarding the CiA 301 and CiA 443 see [8]. Also, [15] is useful complementary reading material.

4.2 IWIS

Introduction

Intelligent well interface (IWIS) is a joint industry project between oil & gas operators, service companies, and downhole equipment manufacturers that evolved in 1995. The initial objective of the IWIS work group was to establish standards for interface of printed circuit boards (PCB) because the selection of downhole gauge vendor was performed a long time after selection of the SCM vendor. This tended to give compatibility problems [13]. At the time there was almost no standardization for subsea equipment, and the scope was therefore expanded to evolve standardisation of physical, electrical, communication and hydraulic interface between intelligent well devices and subsea production control systems [29, 4]. The idea is that a common industrial standard for this purpose will [30]:

- Reduce lead times for hardware (a subsea infrastructure becomes easier to define)
- Reduce direct and indirect cost (easier definitions, wider choice of suppliers, and easier implementation of subsea infrastructure)
- Reduce technical risk (less complexity, minimizing the number of unique interfaces per project)
- Increase flexibility of compatible systems

IWIS also include details for installation of the SEM in regards to mechanical mounting, connectors, electrical limits, redundancy, etc. IWIS specifies requirements for implementing fail-safe functionality by use of hydraulic systems [3].

The IWIS specification was in 2003 submitted to ISO in order to be included as an appendix in the ISO 13628-6. IWIS has contributed to defining physical interfaces such as communication, hydraulic power, control of directional control valve (DCV), electromagnetic compatibility (EMC), tubing hanger feedthroughs & penetrations and procedures for equipment testing. IWIS has also been implemented as an appendix in the American petroleum institute (API) 17F which is the American oil & gas standard for subsea production control systems [4]. Its the communication protocol called IWIS PPP as specified in ISO 13628-6 and in API 17F that will be discussed in this section.

The IWIS Interface

As mentioned above, the joint industry project in 1995 lead to a common industrial standard for interfacing intelligent well equipment (IWE) into a subsea control system. The intelligent well equipment is advance downhole measurement technology which can be used for obtaining real-time process data. The IWIS specification as submitted to the ISO specifies communication, hardware, and power requirements for three different physical options based on where the intelligent well controller (IWC) equipment is physically installed. A communication system architecture of IWIS is illustrated in Figure 4.3. The IWIS interface specification applies to the link between the IWE (both the IWC equipment and sensors & actuators) and the SEM. IWIS also specifies an interface at the surface between the intelligent well control system (IWCS) applications and the subsea production control system (SPCS) by use of TCP/IP and Ethernet. The main idea is that the infrastructure of the subsea production control system (SPCS) from a communication point of view - only provides a transparent transport function between the IWE located subsea, and the IWCS applications at topside [30].

The IWIS PPP communication protocol as described in the standards [4, 20] and the recommended practice [30] specifies the interface between the intelligent well equipment and the SEM. This point-to-point protocol (PPP) is specified according to the OSI reference model as given in Table 4.3. At layer 1 the IWIS PPP is implemented with RS-422 at full duplex, no hardware handshake, and with a default baud rate at 9600 bit/sec [4]. The baud rate should be able to change, but all devices are required to support the default rate at 9600 bit/sec. Omitting to specify application layer details (message structure & formats), and only specify the lower layers of the communication protocol leaves the practical message exchange implementation open & flexible to the vendors of the IWCS. The subsea control system does not require any knowledge about the application protocol used by at the end points (IWCS application and IWE) [4, Section 7.4.4].

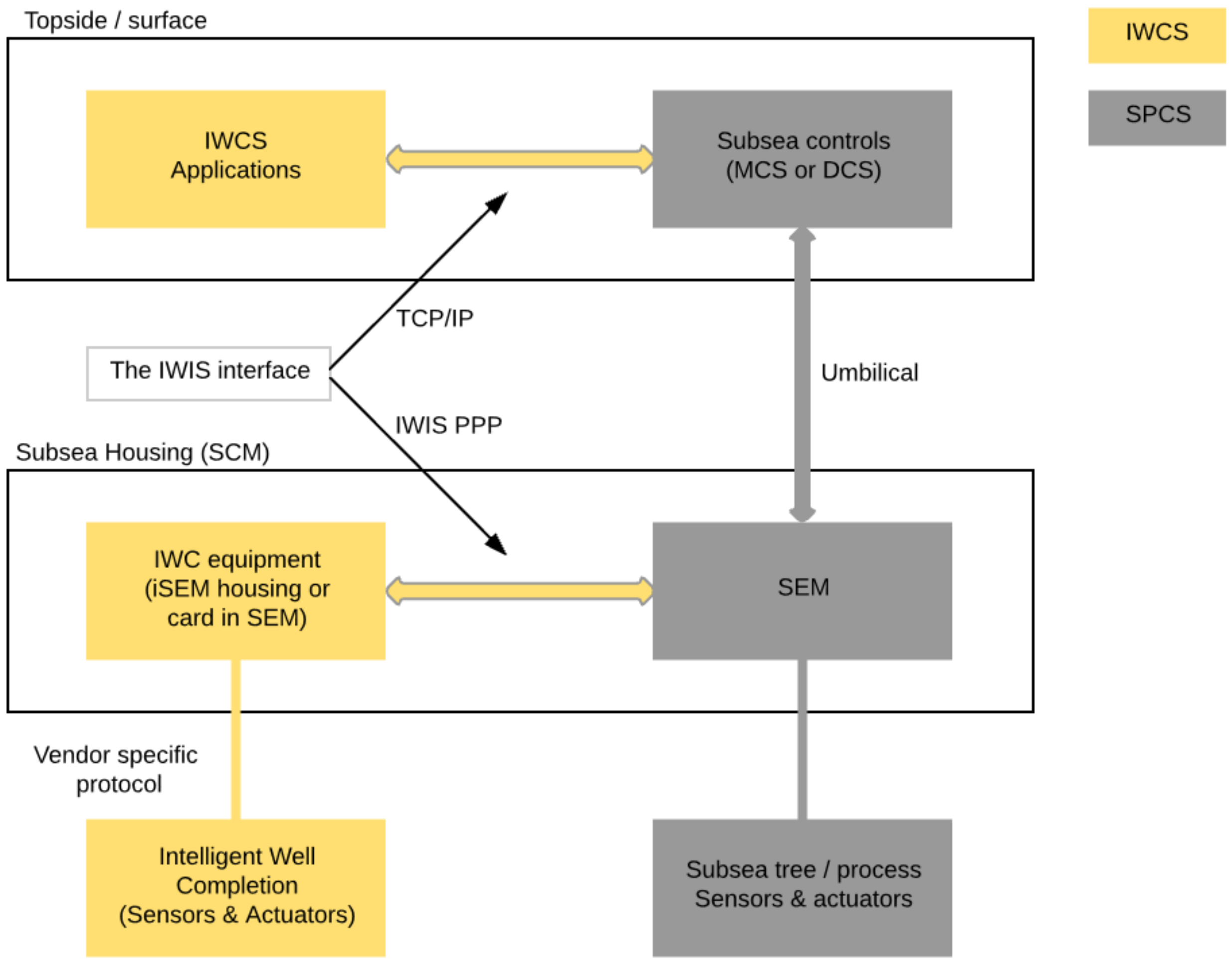

Topside / surface
IWCS
SPCS
IWCS Applications
Subsea controls (MCS or DCS)
TCP/IP
The IWIS interface
IWIS PPP
Umbilical
Subsea Housing (SCM)
IWC equipment (iSEM housing or card in SEM)
SEM
Vendor specific protocol
Intelligent Well Completion (Sensors & Actuators)
Subsea tree / process Sensors & actuators

Figure 4.3: Overview of the production control system & the IWIS. Modernized from [30, Ch. 5]

OSI-model layer	Description
7 - Application	
6 - Presentation	
5 - Session	
4 - Transport	TCP
3 - Network	IP
2 - Data link	Point-to-point protocol (PPP)
1 - Physical	RS-422

Table 4.3: IWIS PPP according to the OSI reference model.

The downhole vendor application located topside, connects via an network access point (NAT) in order to exchange messages between the IWCS applications and the IWE. This makes the whole subsea network operate as a transparent transport function between the downhole vendor application and the IWE [30], as illustrated in Figure 4.4. Its not unusual for the SEM to support multiple IWIS PPP channels, i.e. several downhole controller cards.

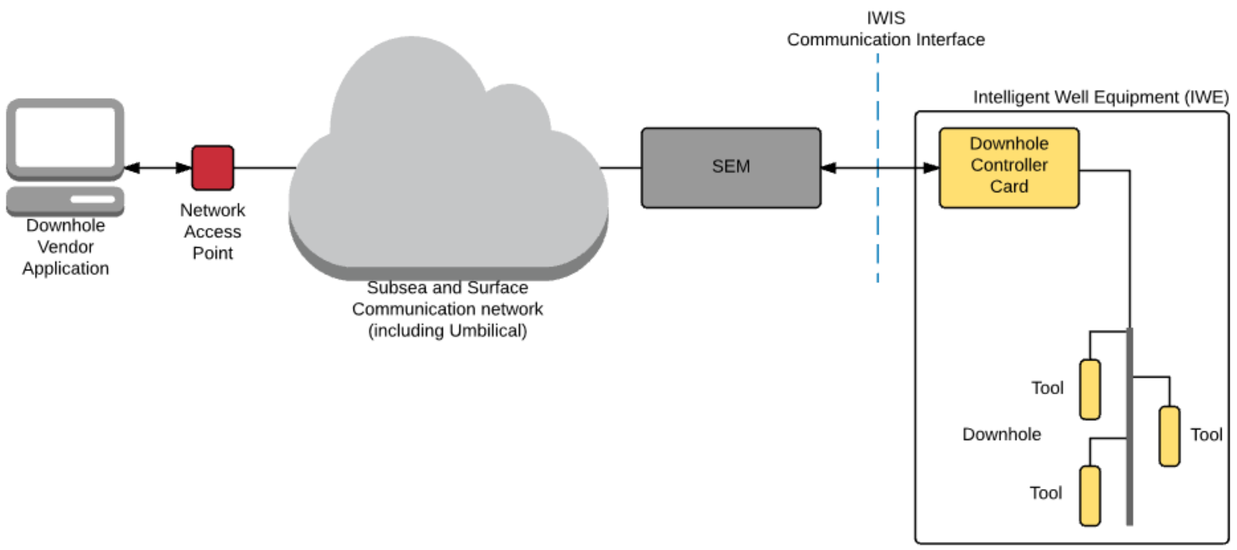

IWIS
Communication Interface
Intelligent Well Equipment (IWE)
Downhole Vendor Application
Network Access Point
SEM
Downhole Controller Card
Subsea and Surface Communication network (including Umbilical)
Tool
Downhole
Tool
Tool

Figure 4.4: The IWIS communication architecture. Modernized from [30, Ch. 7]

Properties of the RS-422

As specified in Table 4.3 RS-422 is chosen at the physical layer of the IWIS interface. RS-422 is a serial, balanced, and differential communication bus. It supports a four wire full-duplex network consisting of one line driver (transmitter) and up to 10 line receivers. However, since the data link layer consist of a point-to-point protocol only one transmitter and one receiver is used. The maximum theoretical length is 1200 meters, and data rates up to 10 Mbit/sec [32, Ch. 3.9]. Similarly as the above mentioned CAN bus it benefits from good noise immunity due to differential voltages.

The great benefit of RS-422 compared to Ethernet as a physical layer is that the RS-422 specification [1] does not define connectors, pin assignments, and functions. It only defines electrical signals for representation of data. As the IWIS specification defines three physical ways of installing the IWC equipment in Figure 4.3, RS-422 is a quite flexible transport media for this purpose. It will support multiple installation methods such as local backplane installation in the SEM, installation within the SCM housing, or an external installation outside the SCM housing.

PPP Operation & Performance

The point-to-point protocol (PPP) as specified in Internet RFC 1661[2] is originally designed as an encapsulation protocol for transporting IP traffic over peer-to-peer connections. The detailed PPP procedure for establish, configure, maintain, and terminate the peer-to-peer connection is given in [30].

As mentioned, IWIS consider the subsea network as a transparent transport function. IWIS does not specify that the link between topside and subsea must use TCP/IP. This may lead to questions regarding timing demands or performance. However, in API 17F [4, F.4.1] a timing requirement for the system response time is set to less than 1 second during normal operation. This shall be performed by carry out a ping-command in the IWCS application.

4.3 Evaluation - Comparison and Discussion

First of all, CAN bus (SIIS level II) is the subsea standard for instrument network. The network is based on a fault tolerant physical layer which uses CANopen, and the device profile CiA 443. IWIS specifies a common industrial interface for connecting third-party intelligent well devices to the subsea production control system. One might ask the question: is there other relevant subsea sensor networks? - That answer could be found by study the thorough decision process behind the standardization of the subsea instrument network.

[1] http://ftp.tiaonline.org/tr-30/TR-30.2/Public/2005%20Meetings/2005-06%20Arlington/For%20Review/TIA-EIA-422-B-Scanned.pdf

[2] https://www.rfc-editor.org/rfc/rfc1661.txt

As mentioned in Section 2.7, SIIS is a industry group consisting of oil & gas operators, service companies, and equipment manufacturers working on standardizing the interface between subsea instrumentation and the SCM. The article [38] describes some of the historical decision process behind choosing a common industrial protocol for subsea instrumentation. A summary of this article follows. The work group started with evolving a reduced list where well understood and established fieldbus standards were considered as sensor & actuator networks based on factors such as:

- Technical suitability
- Acceptability to SIIS members ("cost of ownership")
- Support
- Specification management

This work resulted in a list consisting of three well known standards: CAN, Profibus DP, and Foundation Fieldbus. The work continued with considering other parameters such as the cost of hardware & software for development tools, production licensing, and specific component cost [38]. The SIIS group concluded that CAN was most suited as a subsea instrument network due to several factors including the widely supported physical layer, mass production of components to several types of industries, and its the standard with lowest ownership costs. Also, the CAN bus profited from having a potential fault tolerant feature. Its recommended to read the whole decision process at [38] for understanding why CAN bus is the chosen subsea sensor network with use of the CiA 443 device profile on top of CANopen. Based on the thoroughly decision process performed by the SIIS industry group, I would argue that there are currently not any other subsea instrumentation networks worth mentioning. The CAN bus (SIIS level II) is changing the way subsea instrumentation networks are performed by moving from custom made to more product-based solutions. The CAN bus has gained grounds on the market, and the standard solutions is reducing costs in subsea projects.

As default the IWIS PPP interface has a baud-rate of 9600 bit/sec between the IWE and the subsea production control system. A project specific baud-rate can be agreed, but all devices should support the default baud-rate. The RS-422 transport media is limited to data rates up to 10 Mbit/sec. Subsea control systems tends to operate at 100 Mbit/sec Ethernet on the link between topside and subsea. A default baud-rate at 9600 bit/sec could possible lead the IWIS PPP interface to represent a bottleneck if data exchange between the downhole controller card and the downhole tools (Figure 4.4) is also performed at higher baud rates. However, if acoustic is used at the physical layer between the downhole controller card and the downhole tools, the 9600 bit/sec baud-rate is probably sufficient. Regardless, the potential scale of the bottleneck situation is difficult to estimate, more thorough tests must be performed to determine the extent. An advantageous property of the CAN bus is the detailed documentation of the application layer with the possibility of adding device profiles. An article [13] from FMC regarding implementation experience concludes that the RS-422 interface used for IWIS will probably be replaced by CAN or Ethernet sometime in the future. Replacing the RS-422 physical interface to CAN would make the system more reliable as it decreases the needed hardware components. However, the RS-422 benefits from only specifying the electric characteristics, and are therefore more flexible towards the three different ways to install the intelligent well equipment. This is described in Section 4.2. Only specifying the lower layers of the IWIS PPP allows IWE vendors to engineer proprietary application protocols for message exchange between topside and subsea. The subsea control system does not require any details regarding the application layer, the existing control system infrastructure shall only provide a transparent transport function.

A pure comparison between IWIS PPP and CAN bus (SIIS level II) is challenging because they are engineered for two different purposes. A more realistic scenario would be to compare the instrumentation network used between the downhole controller card and the downhole tools as illustrated in Figure 4.4. This sensor network is typically vendor specific, and is not part of the IWIS scope. However, the instrumentation network for downhole tools are not suited as a CAN bus. The physical layer on CAN bus is based on 2-wires with limited distance & bit-rate between devices. Downhole tools tends to be used inside wells that can be several kilometers long. A communication protocol supporting the use of either acoustic or fiber optical at the physical layer is therefore more practical. Some downhole tools will due to the data amount favour a network with higher bandwidth such as an industrial Ethernet protocol.

CAN bus with its non-destructive bit-wise arbitration gives CAN real-time properties, and offer robust noise immunity due to differential voltages. It has a detailed application layer supporting several device profiles, it has no message collision, and is a low-cost implementation. Experience from test and implementation at FMC Technologies shows that suppliers without CAN experience find the CAN-documentation descriptive and clear. But FMC experiences that they need some coaching for the practical implementation [13]. IWIS is used to connect downhole measurement tools to the subsea production control system. It uses a point-to-point protocol, and a flexible RS-422 transport media. It has a relatively low default baud-rate between the downhole equipment and the SEM. IWIS does not specify how the application-layer for message exchange between topside and subsea should be performed, this can be determined by the individual IWE vendors. Both IWIS PPP and CAN bus (SIIS level II) is a result of joint industry projects (JIP) and serves as an example on the willingness to achieve standardization within subsea. The introduction of both these standards is resulting in reduced lead time, increased interoperability, and steps towards achieving plug and play solutions.

Chapter 5

Practical

5.1 Modbus TCP Implementation

5.1.1 Hardware

The Arduino Due Microcontroller Board

The Arduino Due microcontroller board was chosen as the base hardware for implementation of the Modbus TCP application protocol on Ethernet controllers. The reason for this was mainly due to its easy accessibility, and not the demand for such a powerful microcontroller. The Arduino Due board is based on the Atmel SAM3X8E ARM Cortex-M3 CPU which is a 32-bit ARM core microcontroller. It has extensive support for peripherals including 54 digital I/O pins, 12 analog inputs, 4 UARTs, 2 digital-to-analog outputs, a SPI header, a JTAG header, and CAN-bus interface (CAN TX/RX).

A 32-bit core enables operations on 4 bytes wide data within a single CPU clock. The CPU clock on the Arduino Due operates on 84 MHz, making it relatively powerful compared to other Arduino microcontroller boards such as the Arduino Uno which operates at 16 MHz. The Arduino Due has 96 kB of SRAM (two banks of respectively 64 kB and 32 kB), and a flash memory of 512 kB (2 blocks of 256 kB) which is available for user applications [5]. The Arduino Due microcontroller board contains a reduced instruction set computer (RISC) based microcontroller, an ATmega16U2 which is used as a USB-to-Serial converter. Its used to setup a virtual COM port in the software so that a computer can connect to the USB micro programming port. Figure 5.1

depicts the Arduino Due microcontroller board, and illustrates the location of the programming port. An unofficial pinout diagram of the Arduino Due showing the possible interfaces can be found in Appendix B.1. More hardware related information for the Arduino Due microcontroller board can be found at [5].

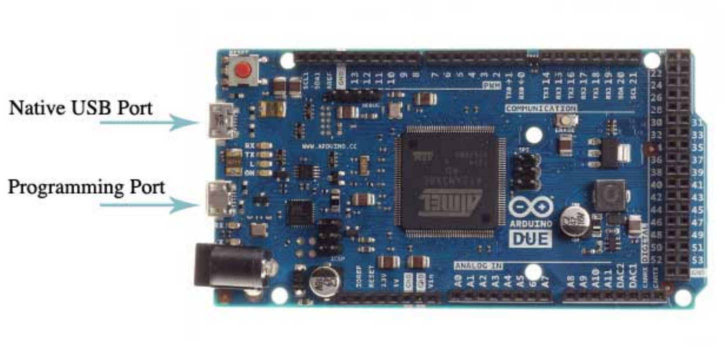


Figure 5.1: The Arduino Due microcontroller board [5].

The Arduino Ethernet Shield

An Arduino microcontroller board can be connected to a Ethernet network by a module called Arduino Ethernet Shield. The Arduino Ethernet Shield contains a Wiznet W5100 Ethernet Controller for embedded systems with a 16 kB data communication buffer. This Ethernet Controller implements the TCP/IP communication stack, and integrates Ethernet according to OSI reference layer 1 & 2. It also supports UDP, IPV4, ICMP, ARP, IGMP, and PPPoE. The Ethernet Controller supports 10/100 Mbit/sec, auto-negotiation at full & half duplex, and auto-MDI/MDIX. Auto-MDI/MDIX is used to detect if the connection requires a crossover (Physically and electrical connected as TX-TX & RX-RX, changes it internally to TX-RX, and RX-TX). It's possible to run four socket connections simultaneous by using this shield [44]. The Arduino Ethernet shield also contains a micro-SD card slot, which can be used to store files for serving over the network. Both the Wiznet W5100 and the SD card connects to the Arduino microcontroller board by using the SPI bus. The Arduino Ethernet Shield is depicted in Figure 5.2. The module also includes LED outputs which signals for TX, RX, full/half duplex, collision, link establishment and speed (10/100 Mbit/sec). Table 5.1 shows the IO-data mapping from the physical pins to the data register of the Arduino Due microcontroller board in this project.

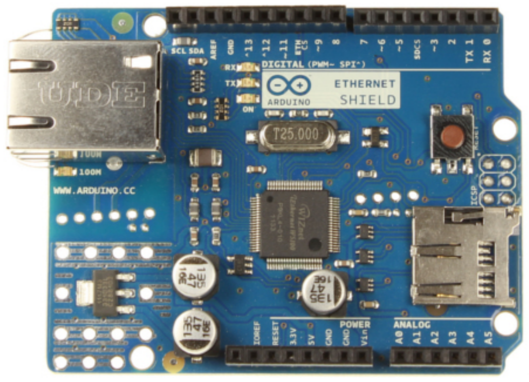


Figure 5.2: The Arduino Ethernet Shield [6].

Physical pin value [1]	**I/O type**	**Memory block type**	**Register address**	**Start address**	**Access level**	**Comments**
54	AI 0	Input register	0	0	Read-only	*10-bit ADC*
55	AI 1	Input register	1	1	Read-only	*10-bit ADC*
56	AI 2	Input register	2	2	Read-only	*10-bit ADC*
57	AI 3	Input register	3	3	Read-only	*10-bit ADC*
58	AI 4	Input register	4	4	Read-only	*10-bit ADC*
59	AI 5	Input register	5	5	Read-only	*10-bit ADC*
60	AI 6	Input register	6	6	Read-only	*10-bit ADC*
61	AI 7	Input register	7	7	Read-only	*10-bit ADC*
62	AI 8	Input register	8	8	Read-only	*10-bit ADC*
63	AI 9	Input register	9	9	Read-only	*10-bit ADC*
64	AI 10	Input register	10	10	Read-only	*10-bit ADC*
65	AI 11	Input register	11	11	Read-only	*10-bit ADC*
		Holding register	12	12	Read/write	*Watchdog counter*
67	AO 1	Holding register	13	13	Read/write	*12-bit DAC*
15 - 0	DI	Discrete inputs	14	224	Read-only	
21 - 16	DI	Discrete inputs	15	240	Read-only	
37 - 22	DO	Discrete outputs (coils)	16	256	Read/write	
53 - 38	DO	Discrete outputs (coils)	17	272	Read/write	

Table 5.1: IO-data mapping for the Modbus TCP server in this project. **Note!** Physical pin 0, 1, 4, 10 & 13 are preconfigured by Arduino and are therefore omitted as inputs in the software.

[1]The physical pin values refers to the pinout diagram in Appendix B.1.

5.1.2 Software

The Modbus Application Layer

This part of the project comprises of implementing an embedded communication system with use of the Modbus TCP protocol. As seen in Table 5.2, the Modbus TCP protocol is defined at layer 1-4 & layer 7 according to the OSI reference model. The Arduino Ethernet Shield integrates the TCP/IP communication stack and provides an Ethernet interface. This reduces the practical assignment to integrate layer 7 - the Modbus application protocol. The application layer is in general terms the layer that user programs and processes access to communicate over the network. It is usually the only access point to users. For a subsea system that uses Modbus TCP communication between topside and subsea, it could be necessary to implement the Modbus application layer at the embedded system located inside the SEM.

Layer	Description
7 - Application	Modbus
6 - Presentation	-
5 - Session	-
4 - Transport	TCP
3 - Network	IP
2 - Data link	Ethernet
1 - Physical	Ethernet

Table 5.2: Modbus TCP related to the OSI model

The implementation of communication, terminology, message format, and general structure of the Modbus application protocol was performed according to the *Modbus application protocol specification* [25] and the *Modbus messaging on TCP/IP implementation guide* [24]. The *Modbus application protocol specification* describes all (approx. 20) of the public Modbus function codes which are applicable for Modbus on serial line. Not all of them are relevant for Modbus TCP. The function codes are related to read/write of single bits and registers (16 bits), diagnostic, exceptions, read/write of file records, CANopen interface, and reading device information (serial. ID, rev. ID). It was decided to implement the relevant functions codes for exchanging process data between the topside controller and the embedded system in the SEM. More specifically, this means function codes in class 0 and 1 [35]. Class 0 refers to the minimum useful

				Function Codes
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	0x02
		Internal Bits Or Physical coils	Read Coils	0x01
			Write Single Coil	0x05
			Write Multiple Coils	0x0F
	16 bits access	Physical Input Registers	Read Input Registers	0x04
		Internal Registers Or Physical Output Registers	Read Holding Registers	0x03
			Write Single Register	0x06
			Write Multiple Register	0x10
			Read/Write Multiple Registers	0x17
			Mask Write Register	0x16
			Read FIFO queue	0x18
	File Record Access		Read File record	0x14
			Write File record	0x15
Diagnostics			Read device identification	0x2B

Table 5.3: Public Function Codes (FC) for Modbus TCP.

set of functions, while class 1 is related to functions that are commonly implemented to ensure interoperability. The implemented function codes are marked grey in Table 5.3. The function codes related to file record access, reading device identification, and reading FIFO queues were omitted (belongs to class 2). Terminology, message format such as client/server, PDU, ADU, and MBAP as described in the introduction of Modbus TCP in Section 3.2.1 are all applied in the software. Further details regarding connection establishments (initiate, maintain and close connections), parametrization, and the MBAP header for Modbus TCP see the *Modbus messaging on TCP/IP implementation guide* [24].

Design Considerations

Below is a list of design considerations that were found relevant for the software implementation of the Modbus application protocol:

- Message addressing between nodes are executed by use of IP-addresses, and port 502 which is reserved for Modbus communication. Both the client and server listens for incoming messages at port 502 [25].

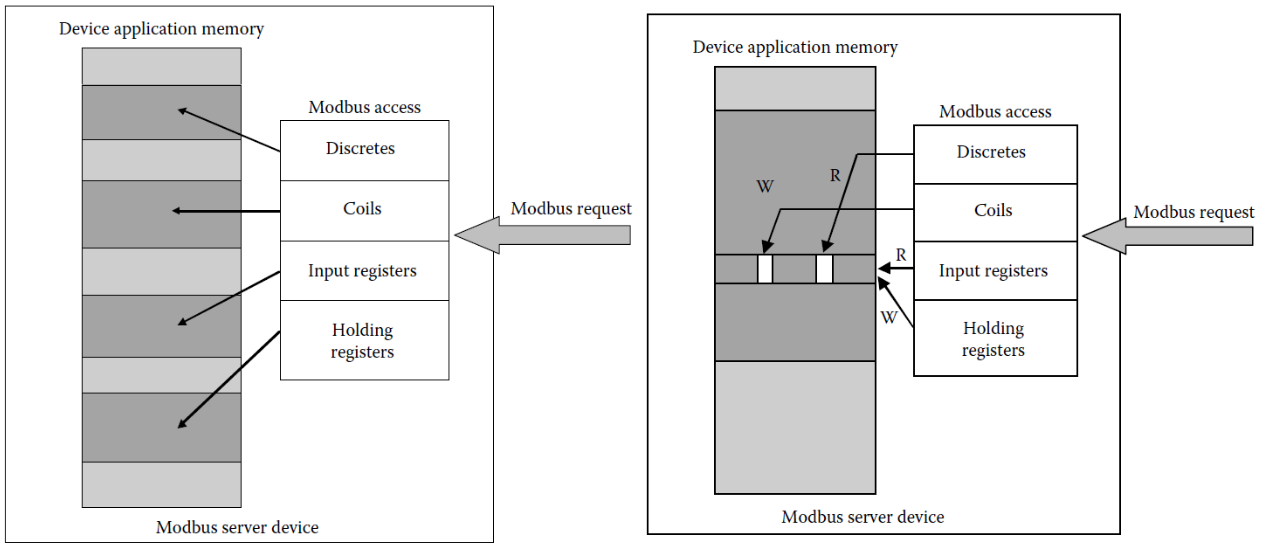


Figure 5.3: Two possible memory block implementation concepts. Memory block solution at the left hand side is called distinct tables because it splits the block into separate sections. The memory block solution at the right hand side collects all data in one section. The Modbus access functions is then overlapping. [46, Ch. 10].

- The maximal message size (ADU) is limited by the Modbus PDU (253 bytes) for serial line communication. This limitation for serial line communication is applicable also for TCP/IP communication because of the potential gateway to serial line Modbus. Ethernet can handle messages of larger size. By including the MBAP header (7 bytes), the maximal ADU for Modbus TCP is limited to 260 bytes.

- The organization of the memory block at the server & client can either be implemented as 4 four separate sections (shown at the left in Figure 5.3) so that access to the single bits and registers can be kept separate. Or, it can be implemented as a single block (right hand side of Figure 5.3) where the same data can be accessed by use of several Modbus function codes. For this project, a design choice was taken, and it was decided to implement the memory block as four separate sections. An overview of the memory block sections for this project can be found in Table 5.1.

- Connection establishments between server & client are performed by use of the TCP/IP sockets. The software drivers of the Ethernet Controller located on the Arduino Ethernet Shield offers possibilities of modifying the timeout and number of retransmission. The default timeout is approximately 32 seconds but for achieving some form of real-time constraint this was modified to 2 retransmissions and a 200 ms timeout.

- Validation of function codes, data addresses, and data values are implemented as Modbus exception response. A deviation from the Modbus exception handling described in the specification [25] is that exception code 4 is not implemented. This is an exception related to errors arising during read/write of data to the memory block. The Arduino IDE disables exception handling so a *try-catch* implementation wouldn't work. The reason for omitting exception handling is the large stack size. Its therefore assumed that when the function code, data address, and data values are within range that the memory block read/write operations is successful. The exception response as implemented in the software, can be seen in Table 5.4. The transaction process when a server first receives a Modbus message is implemented according to the Modbus transaction state diagrams in [25]. Each function code has individual boundary limits for data address and data value.

Table 5.5 shows the ADU request message format sent from the client to the server. Table 5.6 shows the different response ADUs sent from the server after receiving a request ADU from the client. Both the request and response ADU is specified in the *Modbus application protocol specification*. Table 5.4 shows the error response sent to the client when the server receives a message and detects invalid values. The transport protocol (TCP) guarantees that the message sent from the client is identical to the one delivered to the server. The error response mechanism is meant to check for logical errors in the message. When a message first is found invalid its rejected, and not used further. An error response with the structure shown in the table is sent back to the client.

Description	**Function code**	**Exception code**
Function code (FC) invalid	FC + 0x80	0x01
Data values invalid	FC + 0x80	0x03
Data access invalid	FC + 0x80	0x02

Table 5.4: Error response implemented on the server in the embedded communication system

Request/Response ADU Usage Example

An example of ADU usage is when the client requests to *Read Input Registers (0x04)* at the server. The client would then need to send a request ADU according to Table 5.5. The request ADU will consist of a MBAP header (more details in Table 3.2), a function code representing the action to perform (in this case - 0x04), start address of where to find the analog input value (for this implementation see Table 5.1), and the quantity bytes used to described how many registers to read in one Modbus message. The server will then receive a Modbus message, it will decode it and analyze it. If the server does not support the function code, the data values are invalid, or its denied data access it will send an error response in return. This error message will consist of a MBAP header and the selected content in Table 5.4. If the Modbus message passes the validation at the server, it will perform the action in the message, and send a response message. The response message (Table 5.6) will in this case consist of a MBAP header, function code (0x04), a byte count value representing the quantity of registers, and the value(s) found in the register(s).

Description	**MBAP**	**Function code**	**Start/output Address**	**Quantity of:**	**Count**	**Value to write**
Read coils	7 bytes	1 byte (0x01)	2 bytes	2 bytes (coils)		
Read discrete inputs	7 bytes	1 byte (0x02)	2 bytes	2 bytes (inputs)		
Read holding registers	7 bytes	1 byte (0x03)	2 bytes	2 bytes (registers)		
Read input registers	7 bytes	1 byte (0x04)	2 bytes	2 bytes (registers)		
Write single coil	7 bytes	1 byte (0x05)	2 bytes (output)			2 bytes (Output value)
Write single registers	7 bytes	1 byte (0x06)	2 bytes			2 bytes (Register value)
Write multiple coils	7 bytes	1 byte (0x0F)	2 bytes	2 bytes (outputs)	1 byte (N^*)[1]	N^* x 1 byte [1] (Output value)
Write multiple registers	7 bytes	1 byte (0x10)	2 bytes	2 bytes (registers)	1 byte (2 x N^*) [1]	2 x N^+ (Register value)[1]

Table 5.5: Request ADUs implemented on the client in the embedded communication system

[1] $N^* = \frac{\text{Quantity of coils/registers}}{8}$, if remainder $\neq 0 \rightarrow N^* = N^* + 1$. N^+ = Quantity of registers (1 register = 16 bit)

Description	**MBAP**	**Function code**	**Byte count**	**Coil status**	**Register value**
Read coils	7 bytes	1 byte (0x01)	1 byte (N*) [1]	n byte (n = N or n = N+1)	
Read discrete inputs	7 bytes	1 byte (0x02)	1 byte (N*) [1]	n byte (n = N or n = N+1)	
Read holding registers	7 bytes	1 byte (0x03)	1 byte (2 x N^+)[1]		(2 x N^+)[1] bytes
Read input registers	7 bytes	1 byte (0x04)	1 byte (2 x N^+)[1]		(2 x N^+)[1] bytes
Write single coil	7 bytes	1 byte (0x05)		2 bytes (Output address)	2 bytes (Output value)
Write single registers	7 bytes	1 byte (0x06)		2 bytes (Register address)	2 bytes (Register value)
Write multiple coils	7 bytes	1 byte (0x0F)		2 bytes (Start address)	2 bytes (Quantity of outputs)
Write multiple registers	7 bytes	1 byte (0x10)		2 bytes (Start address)	2 bytes (Quantity of registers)

Table 5.6: Response ADUs implemented on the server in the embedded communication system

5.1.3 Communication Test

Class diagrams illustrating the attributes & methods for the Modbus TCP application layer can be found in Appendix B.3. The communication setup for the Modbus TCP implementation has the same architecture as in Figure 3.1. In a subsea project, the architecture would likely consist of a redundant client at topside exchanging data with one or several SCMs (servers). For demonstration purposes a Modbus client using several of the request messages was implemented as the user defined layer. The client periodically sends: watchdog counter messages, LED on/off messages, and request message to read an analog input value (input register). The sequence diagram showing the operation for both the Modbus client and server can be found in Appendix B.2. The Modbus user-defined layer and library for both the Modbus TCP client and server can be found in Appendix C.1.1.

[1] $N^* = \frac{\text{Quantity of coils/registers}}{8}$, if remainder $\neq 0 \rightarrow N^* = N^* + 1$. N^+ = Quantity of registers (1 register = 16 bit)

Modbus Client

The Modbus client is implemented as a multi-task client with FreeRTOS as an operating system. Its assumed that the reader is familiar with FreeRTOS basics. The client has three tasks as mentioned above, see Figure B.2 in Appendix B.2 for further details. The first task periodically sends out *Write Single Coil (0x05)* messages to the client for turning the external LED on/off, and request the input register value at the server using the *Read Input Register (0x03)* message. The second task listens & process incoming confirmation messages from the server. The third task is the watchdog function. It sends *Write Single Register (0x06)* messages to the server, waits, and continues if a correct counter value was received before the next cycle. The idea of the watchdog is an execution test of the server, if the server is answering correctly on the message - the server is operating correctly. If the server do not reply or replies with the wrong value, the watchdog will put the client in a safe state. Both the first and third tasks uses the client send function, to ensure exclusive access a mutex called *sendMutex* is used, this can be seen in the class diagram in Appendix B.3. In a real subsea system the client would probably be a single task controller while in this task its a multitask system. The reason for choosing a multitask system for this setup is to achieve a better microcontroller performance, and a more logical software division.

Modbus Server

The Modbus server is also implemented as a multi-task server with FreeRTOS as an operating system. The Modbus server has two tasks, see Figure B.3 in Appendix B.2. The first task periodically reads and writes IO data from physical pins to the server data registers. The same task also sets two LEDs on the Arduino Due microcontroller board on/off during this period for control check purposes. The second task listens for incoming Modbus messages, processes them, and replies to the client. Both of the tasks reads and writes data to the server register data so a mutex (*rwMutex*) is used to avoid race condition.

In a real subsea system its not improbable that the server is a multitask system, where one task handles communication & processing, and another task handles mapping of IO into the memory block. These task tends to run with various time periods.

5.2 OPC UA Implementation

5.2.1 Hardware

The Raspberry Pi 2 Model B

The initial plan for the OPC UA implementation in this project was to use the same hardware as for the implementation of the Modbus TCP protocol, the Arduino Due microcontroller board. However, after reaching out to OPC software developers it became clear that they did not currently support the CPU architecture for the Arduino Due microcontroller board, the ARM Cortex M3. It was recommended to use a Raspberry Pi (RPi) as hardware for the implementation of either an UA server, an UA client, or an UA server/client as an architectural model. Software developers tends to support the Raspberry Pi since its a quite popular developer tool for their OPC UA software development kits. For this part of the project the Raspberry Pi 2 model B is used. This is the second generation Raspberry Pi microcomputer. The technical specification for the Raspberry Pi 2 model B can be found in Table 5.7. Figure 5.4 depicts the Raspberry Pi 2 model B.

	Raspberry Pi 2 Model B
System-on-a-chip (SoC) (CPU, GPU, DSP, SDRAM and USB port)	Broadcom BCM 2836
Central processing unit (CPU)	900 MHz quad-core ARM Cortex-A7
Graphics processing unit (GPU)	VideoCore IV (3D graphics core)
Memory (SDRAM)	1 GB
USB ports	4 USB 2.0
Video & audio output	HDMI & combined 3.5 mm audio-jack, and composite video
Onboard storage	MicroSD card slot
Onboard network	10/100 wired Ethernet port (RJ45)
Low-level peripherals	40 GPIO pins
Other interfaces	Camera Interface (CSI) & Display interface (DSI)
Size	85 x 56 x 17.0 mm
Power rating	Approx. 650 mA (3.0 W)
Weight	45 g

Table 5.7: Raspberry Pi 2 model B specifications [34].

The general purpose input/output (GPIO) pins on the Raspberry Pi provides a physical interface between the RPi and other low-level peripherals. It can be used for digital input and outputs, interrupt sources to the ARM CPU, SPI, I^{2}C, and serial UART. Internal pull-up / pull-down resistors can also be enabled/disabled. A 16 GB class 10 MicroSD memory card was used in this project.

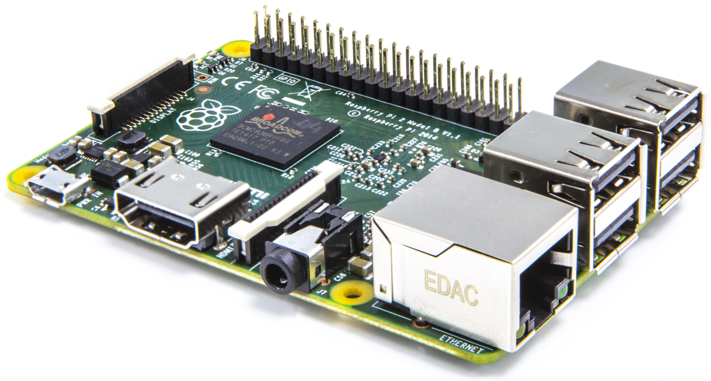


Figure 5.4: The Raspberry Pi 2 model B [34].

Operating Systems

The Raspberry Pi is delivered without a basic input-output system (BIOS). An operating system must be flashed onto a micro secure digital (MicroSD) card that is inserted into the card reader on the RPi. More details regarding the installation of the OS is given below in Section 5.2.3. The Raspberry Pi Foundation offers the Debian based Raspbian operating system for free download. There are also several third party operating systems available for the Raspberry Pi. Ubuntu, and Windows 10 Internet of Things (IoT) core are two of them. An overview of the supported operating systems can be found at [1].

The OPC UA SDK used for this project only supports the Windows 10 IoT Core operating system. The Windows 10 IoT Core is a optimized version of the Windows 10 designed for small embedded devices such as the Raspberry Pi. The hardware requirements are a processor running at 400 MHz or faster, can store 2 GB or more, and has minimum 256 MB RAM. The Windows 10 IoT Core is based on the Universal Windows Platform (UWP) API. More information regarding the Windows 10 IoT Core and developer samples can be found at [2].

[1] https://www.raspberrypi.org/downloads/
[2] https://developer.microsoft.com/en-us/windows/iot

5.2.2 Introduction to the OPC UA SDK

The C/C++ OPC UA server/client software development kit (SDK) bundle used in this thesis was obtained by sending a request to Unified Automation in Germany. They distribute an evaluation edition of the C/C++ OPC UA SDK supporting embedded systems such as the Raspberry Pi 1 & 2, and the Beagle Bone Black. An evaluation edition means it contains examples & tutorials with time limited execution. The server SDK or the client SDK will run for one hour and afterwards it needs a restart to continue the execution. The OPC UA SDK is a C/C++ collection of libraries, helper functions, security mechanisms, documentation, and sample code that support users in writing portable OPC UA servers and clients. Both the server and client uses the same base library which encapsulate the communication stack defined by the OPC Foundation.

The evaluation SDK supports data access, alarms & conditions, and historical data access. The first lesson in the OPC UA SDK is to setup a basic server console application for connection purposes. The basic console performs a global initialization of the OPC UA stack, start-up and shutdown commands, and initialization of the standard server object. The tutorials then proceeds with describing object models that defines the OPC UA address space for real world systems. The SDK also contains support for adding methods and events. The alarms & conditions and historical data access functionality can also be implemented. This modular structure illustrates how scalable the OPC UA is by enabling the programmer to add or remove desired functionality based on the available resources.

A coarse architectural overview of the OPC UA implementation can be described by functions divided in three levels. The SDK provides higher level functions such as the OPC UA stack APIs, base & common UA functionality, helper functions, and security handling. The OPC UA communication stack as defined by the OPC Foundation provides the lower level functions, and are common to both the client and server. The OPC UA communication stack implements the platform, transport, security, and encoding layers for message exchange between different UA applications. The client and server applications are an user-case specific layer on top of the SDK. The application layer is used for application specific logic. This coarse architectural overview is depicted in Figure 5.5. More OPC information regarding services, layers and OPC UA SDKs can

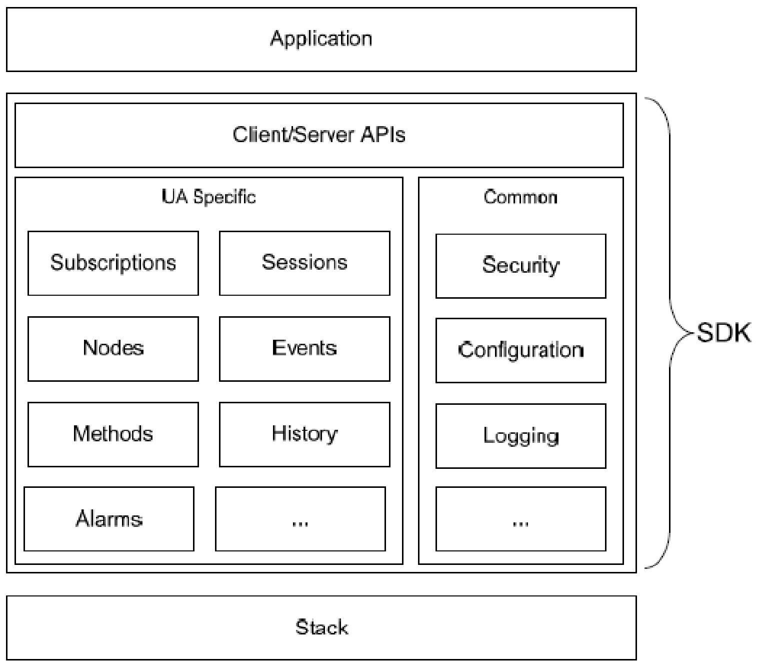


Figure 5.5: OPC UA SDK overview [22, Ch. 8.4].

be found in [22]. For more information regarding each respective SDK, see the installed product documentation. The documentation for the SDK used in this project is included in the *doc* folder in Appendix C.2.1.

5.2.3 Software

Installation

A prerequisite for using the OPC UA SDK from Unified Automation is the use of a Windows 10 host computer with Visual Studio 2015 as build environment (more details in the readme file in Appendix C.2.1). A setup procedure of the host computer is provided in Appendix A.1.1. The OPC UA SDK contains *CMakeFiles* that must be built using the CMake software. Hardware requirements, and a detailed build & installation guide of CMake, the Windows 10 IoT Core and the OPC UA SDK on the Raspberry Pi can be found in Appendix A.1.
After building the OPC UA SDK for Raspberry Pi using the CMake software on a host computer, an ARM cross-tool was used to cross-compile a binary suited for the Raspberry Pi (RPi) in Visual Studio 2015. Some common library files & binary files were transferred to the RPi using ftp, and then executed using remote commands from the host computer. Cross-compiling is normally

performed when the target has limited system resources. The RPi is however relatively powerful compared to other microcomputers, but it has limited RAM resources (ref. Table 5.7) for managing the build operation, and is slow compared to an average desktop computer. Also, in this project the great benefit of cross-compiling from a desktop computer is the use of Visual Studio as development tool.

Powershell is recommended by Unified Automation to be used for executing remote commands on the RPi. There is however a bug in the Windows 10 IoT core, the bug is related to printouts in the PowerShell command window during execution. This means the output of the server, containing Endpoint information including IP-address and port number to connect the OPC UA client is not available when starting it from PowerShell. Description of this bug is given in the *readme_raspberry2_win10IoT.txt* file in Appendix C.2.1. Due to this annoying bug the Putty software was used as a SSH client instead of PowerShell in order to remote start the server.

Model and Access Information

UaModeler [1] from Unified Automation is a helpful software tool for creating illustrative OPC UA information models and code generation. The UaModeler provides a graphical design of the address space (theory explaining address space can be found in Section 3.4). The UaModeler uses a template for generating code that describes the information model. It can be used for creating C/C++ and .NET code for both servers and clients.

In this project an OPC UA server representing the embedded system in a SEM was implemented on the Raspberry Pi. An illustration of the setup is shown in Figure 5.6. The OPC UA server collects process data from subsea sensors and actuators. This is process data that the UA client at topside can access through the OPC UA server's address space. An alternative representation is to implement OPC UA at sensor level, and as mentioned in Section 1.2 research is performed on this topic. However, for subsea usage an OPC UA server at the SEM appears to be a more mature & reliable solution. The two object models described below is used to represent the address space in the OPC UA server. An derived base object type called *ValveType* was developed.

[1] https://www.unified-automation.com/products/development-tools/uamodeler.html

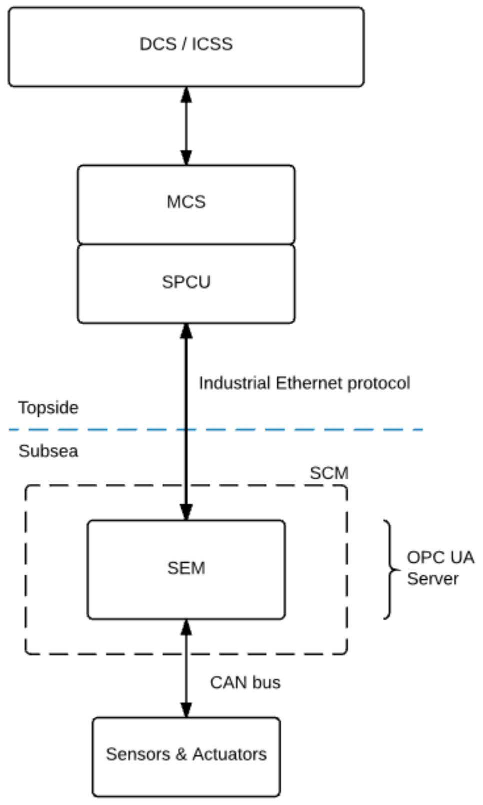


Figure 5.6: OPC UA Server communication setup

This valve object type is defined in terms of variables such as the state, position and setpoint. It also includes call methods for start, stop, and an one-argument method which starts the valve with a given set point. All of the variables and the methods for this valve object type is described with a "HasComponent" reference. Potential references to other objects, would also have been described in this object model. Figure 5.7 illustrates a valve object, *SubseaValve* using this valve object type. The OPC UA server is also implemented with event functions (blue section) for the *State* and *ValvePosition*. The valve object also supports historical data access (green section) for the valve position and the valve set point as depicted in the figure. The server will also generate an alarm if the state is set to zero, representing that the valve is stopped/deactivated. The *ValveSetPoint* is the only variable enabled with read & write access, the two others only support read access. The write access of the set point enables change of the valve opening from the OPC UA client located topside. From the OPC UA client it will be possible to browse and see the details regarding the object type (*ValveType*), and values for the attributes in the valve object (*SubseaValve*).

The second object model describes the rest of the address space which contains four instruments, two temperature and two pressure transmitters, all depicted in Figure 5.8. These transmitters are an *AnalogItemType* which is a predefined OPC UA SDK type derived from an *DataItemType*. The *AnalogItemType* is an object representing analog inputs, it also includes predefined properties such as engineering ranges and engineering units. The transmitters represents inputs to the SEM based at one of the SIIS device levels.

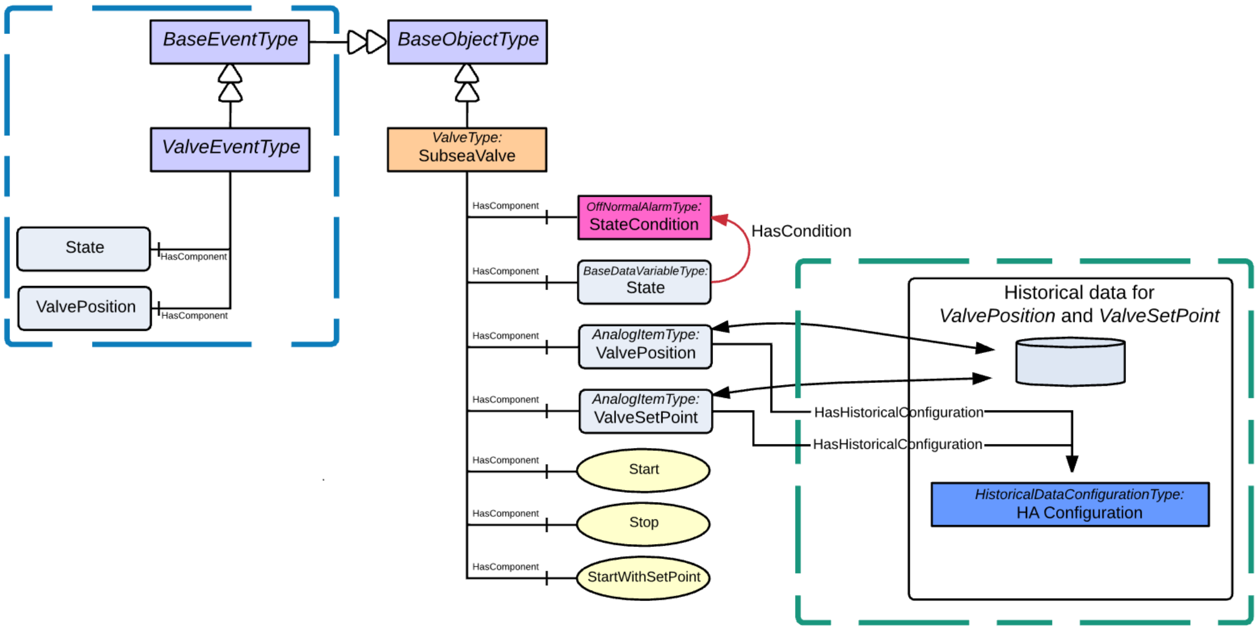


Figure 5.7: Graphical representation of the object model (1 of 2) at the OPC UA server.

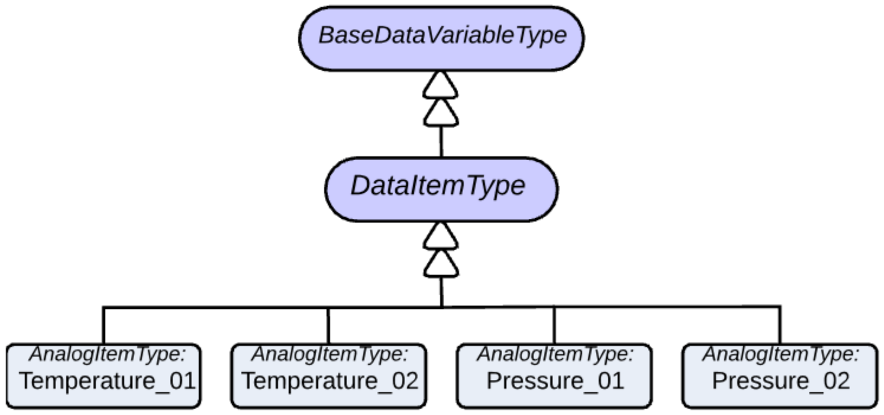


Figure 5.8: Graphical representation of the object model (2 of 2) at the OPC UA server.

Communication Testing

The two object models in Figure 5.7 and 5.8 describes the OPC UA address space for our real world system using UA terminology. These object models was afterwards implemented into the OPC UA SDK, and the project file solution can be found in Appendix C.2.2. Figure 5.9 is an UML diagram illustrating the relationship between the interfaces in the SDK, however some classes related to the simulation of valve data is not considered relevant and are therefore not present. The green color represents interfaces for different node classes as defined by the OPC UA specification, and the yellow color are implementation of these interfaces as found in the SDK. The classes with white color is user defined and implemented into the SDK as part of this project. More information regarding the classes and interfaces in the OPC UA SDK can be found in Appendix C.2.1.

During testing, both the OPC UA server and client were connected in the same local network. The *UaExpert* software [40] was used as an OPC UA client on the host computer. The setup of the OPC UA communication model is comparable to the setup of the Modbus TCP communication model in Section 5.1 with a server and a client. The RPi represents the embedded server system in the SEM, and the client represents the topside controller in a subsea communication architecture. By using the UaExpert as a client the OPC UA server was browsed for objects, variables, events, alarms and historical trending. Figure 5.10 depicts the implementation result in the UaExpert software, as defined by the object models in Figure 5.7 and 5.8. *Note* that several subsea valve objects was declared and instantiated, and contains all the variables, methods, event data, alarm, and historical data access (HA Configuration) from Figure 5.7. The *ValvePosition* and *ValveSetPoint* samples data each 500 ms into a FIFO-queue holding 2000 elements at the server.

5.3 Experiences, Results and Discussion

Section 5.1 documents the practical implementation of the Modbus TCP application layer on two microcontrollers using request/response messages in a server-client architecture. The focus was to implement the main function codes for accessing bits and registers (class 0 and 1). The implementation was performed according to the Modbus specifications [25, 24]. A multi-

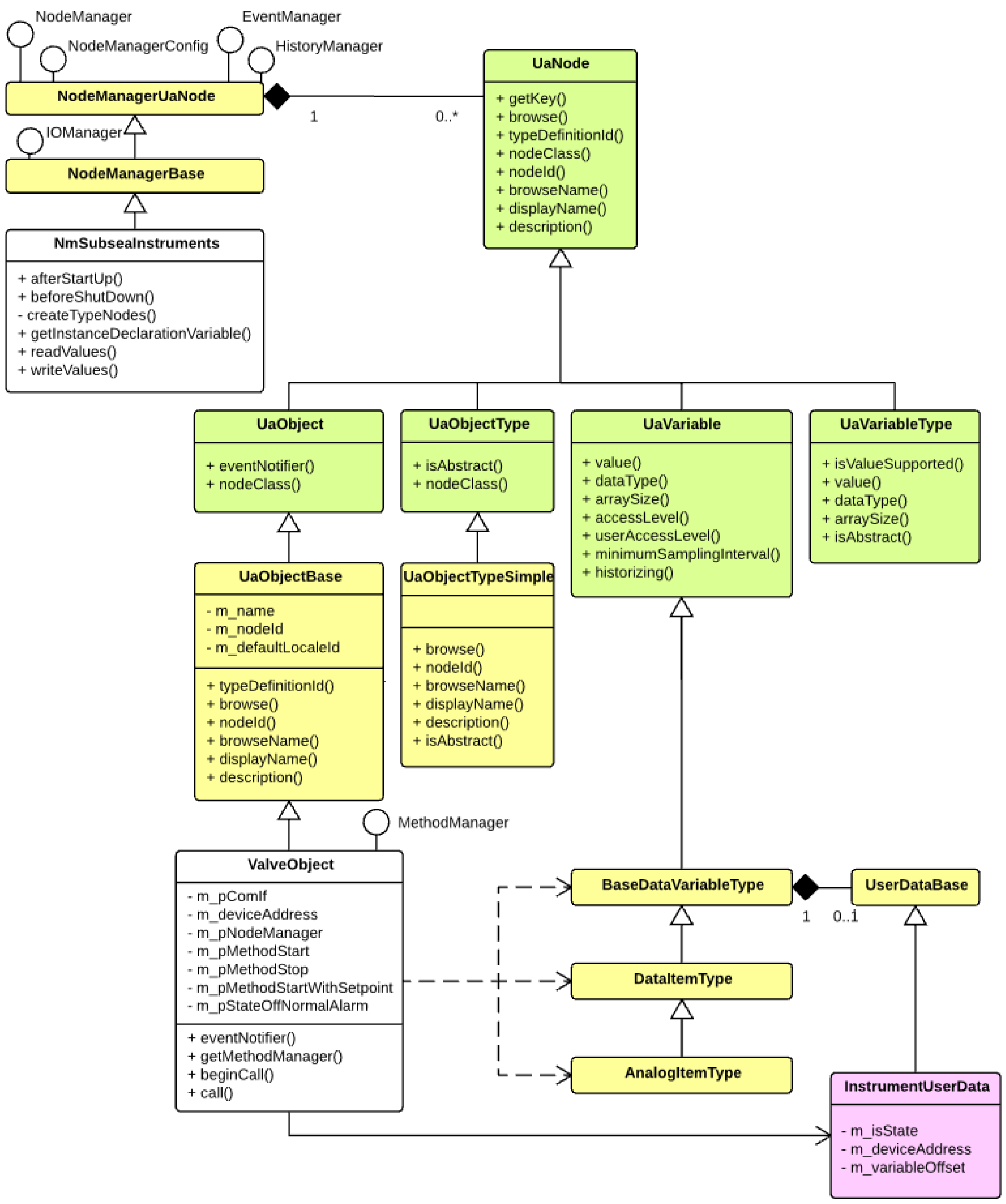
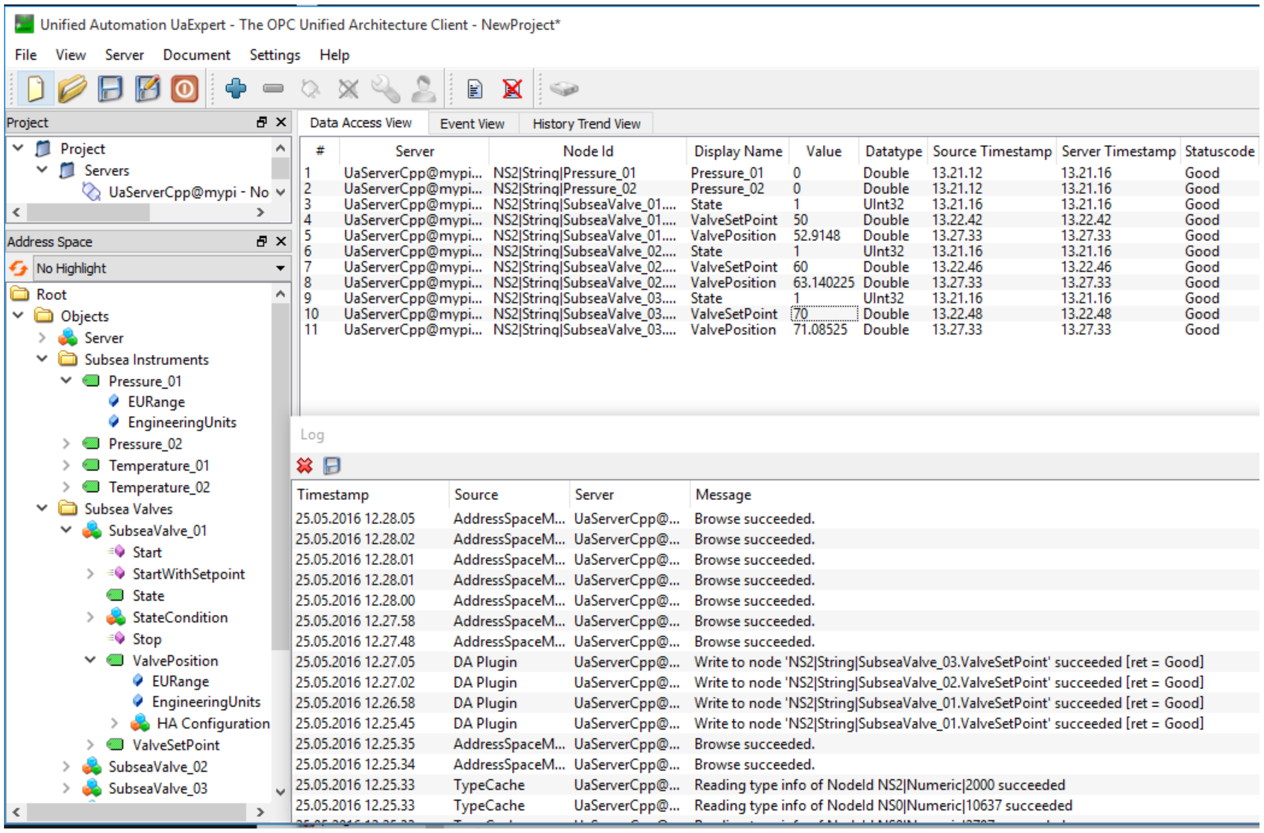


Figure 5.9: UML diagram illustrating the classes and interfaces in the OPC UA SDK.

task server was setup using FreeRTOS as an operating system with two tasks, one task that read & write IO data to the memory block, and the other task for communication exchange. The



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	UaServerCpp@mypi...	NS2|String|Pressure_01	Pressure_01	0	Double	13.21.12	13.21.16	Good
2	UaServerCpp@mypi...	NS2|String|Pressure_02	Pressure_02	0	Double	13.21.12	13.21.16	Good
3	UaServerCpp@mypi...	NS2|String|SubseaValve_01....	State	1	UInt32	13.21.16	13.21.16	Good
4	UaServerCpp@mypi...	NS2|String|SubseaValve_01....	ValveSetPoint	50	Double	13.22.42	13.22.42	Good
5	UaServerCpp@mypi...	NS2|String|SubseaValve_01....	ValvePosition	52.9148	Double	13.27.33	13.27.33	Good
6	UaServerCpp@mypi...	NS2|String|SubseaValve_02....	State	1	UInt32	13.21.16	13.21.16	Good
7	UaServerCpp@mypi...	NS2|String|SubseaValve_02....	ValveSetPoint	60	Double	13.22.46	13.22.46	Good
8	UaServerCpp@mypi...	NS2|String|SubseaValve_02....	ValvePosition	63.140225	Double	13.27.33	13.27.33	Good
9	UaServerCpp@mypi...	NS2|String|SubseaValve_03....	State	1	UInt32	13.21.16	13.21.16	Good
10	UaServerCpp@mypi...	NS2|String|SubseaValve_03....	ValveSetPoint	70	Double	13.22.48	13.22.48	Good
11	UaServerCpp@mypi...	NS2|String|SubseaValve_03....	ValvePosition	71.08525	Double	13.27.33	13.27.33	Good

```
Timestamp            Source            Server           Message
25.05.2016 12.28.05  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.28.02  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.28.01  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.28.01  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.28.00  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.27.58  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.27.48  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.27.05  DA Plugin         UaServerCpp@...  Write to node 'NS2|String|SubseaValve_03.ValveSetPoint' succeeded [ret = Good]
25.05.2016 12.27.02  DA Plugin         UaServerCpp@...  Write to node 'NS2|String|SubseaValve_02.ValveSetPoint' succeeded [ret = Good]
25.05.2016 12.26.58  DA Plugin         UaServerCpp@...  Write to node 'NS2|String|SubseaValve_01.ValveSetPoint' succeeded [ret = Good]
25.05.2016 12.25.45  DA Plugin         UaServerCpp@...  Write to node 'NS2|String|SubseaValve_01.ValveSetPoint' succeeded [ret = Good]
25.05.2016 12.25.35  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.25.34  AddressSpaceM...  UaServerCpp@...  Browse succeeded.
25.05.2016 12.25.33  TypeCache         UaServerCpp@...  Reading type info of NodeId NS2|Numeric|2000 succeeded
25.05.2016 12.25.33  TypeCache         UaServerCpp@...  Reading type info of NodeId NS0|Numeric|10637 succeeded
```

Figure 5.10: Successful browsing of the OPC UA server using the UaExpert software as UA client.

project files containing the implementation of the Modbus application layer for both the client and server can be found in Appendix C.1.1.

Section 5.2 describes the implementation of a platform independent OPC UA SDK solution. This SDK depends only on the OPC UA Communication stack as specified by the OPC Foundation. As mentioned in Section 3.4, OPC UA offers extended possibilities for security measures. The SDK contains tutorials explaining the security mechanisms such as user authentication, message encryption, and certificates but these has not been given much consideration in this project. More documentation and tutorials regarding security mechanism in OPC UA can be found in Appendix C.2.1.

The OPC UA solution serves as an example for how a prospective implementation of OPC UA can be performed in a subsea communication network. The solution recommends keeping the CAN bus as sensor network, and not use OPC UA at sensor level. As mentioned in Section 1.2

some research is performed on the implementation of UA at sensor level. Due to the scalability of OPC UA such a solution should be theoretically possible, however the technology is not considered mature enough. Also, Section 3.5 discusses a research study on an OPC UA SDK which reveals poor network load due to inefficient internal architecture. A CAN network will be advantageous due to the good real-time and noise immunity properties. Sensor & actuator data to/from the CAN bus will be modelled in the UA server address space.

For OPC UA, the main focus has been to understand how to model and access information as defined in the specifications provided by the OPC Foundation [28]. Part 3 of the OPC UA specification provides detailed descriptions on how an address space within an OPC UA server is built and managed. The client uses the sophisticated services defined in Part 4 of the OPC UA specification to access information provided by the server in the UA address space. This is services related to access objects with read or write of an attribute, calling a method or receiving events from the object. The SDK has also made concepts & terms related to OPC UA quite familiar, such as address space, node model & node classes, references, views, event types, services (discovery, query, node, method, subscription) and various types of managers (node, event, history).

The practical experience of implementing Modbus TCP and OPC UA has revealed that UA is a much more complex and abstract communication protocol. However, it represents new possibilities due to its modelling capabilities. The primary focus of the UA implementation was to define an information model, therefore creating object models such as illustrated in Figure 5.7 and 5.8. Data exchange is performed by sending objects with all its parameters, and not sending specific value updates (value changes, alarms and events) [37]. The focus during the Modbus TCP implementation was getting the "bits & bytes" of the message structure in the send and receive functions correct. This was quite the opposite for OPC UA where predefined services is used for communication exchange in the SDK. A great disadvantage of Modbus TCP is that it does not support object oriented modelling. Hence, the Modbus client needs to know exactly which register addresses the specific data is located at the server. Table 5.1 documents the register addresses for the Modbus TCP server in this project. Ethernet/IP and OPC UA supports object modelling.

My experience of Modbus TCP based on the practical implementation in this project is that Modbus is a quite useful approach for data exchange with a specific set of defined function messages. For applications without any safety- or time critical demands its easy to see why the usage of Modbus TCP is so widespread and popular! Even though Section 3.5 reveals poor real-time properties and performance. Modbus does not require any comprehensive SDK, and is easy to integrate on an embedded system. OPC UA appears to be a more elegant way of exchanging data, but for embedded systems where the resources tend to be limited Modbus TCP seems to a better candidate. Modbus has less message overhead & footprint, its easier to debug, and probably has the best application layer efficiency. An example on good message efficiency for Modbus TCP are the function codes for read or write of multiple registers (see Table 5.3), several register values can be transmitted in one request/response frame. This results in a high payload.

The extended use of object modelling in OPC UA enables software developers to use object modelling tools such as the UaModeler to develop a graphical overview of complex components, sub-systems and systems. The development tool will then generate source code for OPC UA that describes the system. For subsea components it could be possible to develop objects for pumps, meters, compressors, separators, tree's, etc in a library. As mentioned in Section 2.7 the MDIS work group has chosen OPC UA as the unified platform between the DCS and the MCS. MDIS has also developed subsea objects for this purpose.

Chapter 6

Discussion and Recommendations for Further Work

6.1 Discussion

A brief introduction to each of the three industrial Ethernet protocols intended for the topside - subsea link are given in Section 3.2 - 3.4, followed by a theoretical evaluation in Section 3.5. The evaluation focuses on performance and available features. The evaluation section has only partly been successful in obtaining relevant quantified comparison data for the industrial Ethernet protocols performed by neutral independent third parties. It seems like comparisons of industrial protocols are typically performed by coarsely dividing parameters into groups (little - medium - large) which provides rough estimates. The main reason for this is probably because the protocols will have far too varying performance based on the hardware, network, and application layer implementation efficiency. In particular this is thought to apply for protocols such as Modbus TCP and OPC UA which are scalable & based on the TCP/IP communication stack and the Ethernet standard. Performance indicators for Ethernet/IP has been obtained from relevant standards. One of the initial objectives of this thesis was to include a detailed benchmark comparison by implementing the protocols on the same hardware and network. Due to unreasonable costs for two FPGA development kits including all needed protocols and software this was omitted. This has resulted in lack of test results that could have been used to back the dissimilarities from the theoretical evaluation in Section 3.5. A benchmark comparison based on the two practical implementations performed in this thesis could have been performed, but such a comparison was not considered relevant due to the large variations in hardware.

The most relevant performance comparison of Modbus TCP and Ethernet/IP collected in this thesis can be found in Figure 3.11 in the evaluation Section 3.5. The comparison of the minimum cycle-time reveals that Ethernet/IP is superior to Modbus TCP in a multicast communication setup operating at 10 Mbit/sec, and also in some degree at 100 Mbit/sec. However, as seen in Figure 3.11 a considerable number of servers (SEMs) must be implemented for any significant variations in the performance. Also, notice that the performance difference between Modbus TCP and Ethernet/IP changes considerably from 10 Mbit/sec to 100 Mbit/sec. The topside - subsea link tends to operate at 100 Mbit/sec Ethernet, but the range of industrial communication equipment supporting 1 Gbit/sec is expanding. A bandwidth of 1 Gbit/sec will probably make the performance differences between communication protocols negligible, and the bottleneck will probably turn out to be the processing power of the end devices.

A brief discussion on the principle of time/event-triggered communication is performed in Section 2.8. The choice of implementing either time or event-triggered communication will depend on the specific application. However, the focus was to highlight the main differences. For subsea communication its recommend to use time-triggered communication because of its predictability, and its characteristic to easily detect missing messages.

Chapter 4 introduces IWIS and the CAN bus (SIIS level II). The initial plan was to compare IWIS and CAN bus (SIIS level II) based on network criterion such as half/full duplex, bandwidth, throughput, and general performance. However, the problem is that IWIS only defines an interface that intelligent well equipment uses to integrate into a subsea production control system. The obtainable performance by using the IWIS interface is too dependent on the vendor specific intelligent well equipment. The evaluation section thereby concludes that a direct comparison of these two principles were not relevant as they are engineered for two different purposes. A more topical comparison would have been to compare the instrumentation network between the downhole controller card and the downhole tools as illustrated in Figure 4.4 against another instrumentation networks. However, CAN bus is not suited as instrumentation network for downhole tools due to its 2-wire physical layer. Regardless, the instrumentation network for downhole tools is not part of the IWIS specification scope, and is regarded as vendor specific which means there exists a wide range of solutions. The evaluation part of Section 4.3 therefore focus on highlight implementation experience from the industry, identify challenges, explaining the selection process for establishing CAN bus as *the* standard for subsea instrumentation, and discuss the physical RS-422 transport media for IWIS. Lack of relevant literature regarding IWIS and the CAN bus has also partially halted the work, most of the information has been found in standards and recommended practices. My experience is that the practical information is still a trade secret, even though both of the principles urges for open standardization. CAN bus (SIIS level II) is detailed specified at higher and lower layers according to Table 4.2, while IWIS is only defined at lower layers according to Table 4.3. The IWIS specification does not state how the message exchange between the IWCS applications and the IWE shall be performed. This provides IWE vendors the flexibility to select/develop their own application protocols for the message exchange. Hence, such proprietary solutions makes further evaluation challenging. Parameters such as protocol overhead and payload efficiency will probably vary between the different solutions. Also, access to the proprietary specifications could also present a challenge for further work.

The OPC UA solution illustrated in Figure 5.6 serves as an example for how a prospective implementation of OPC UA can be performed in a subsea communication network. The solution recommends keeping the CAN bus as sensor network due to its real-time and noise immunity properties. A solution involving OPC UA at sensor level does not seem suited for this purpose, and the technology does not seem mature enough to handle it yet. An example is the paper [17] that reveals poor network load utilization due to inefficient internal architecture for a Java-based OPC UA SDK. The UA solution demonstrates how subsea instrumentation data is implemented into the address space on an UA server. The primary focus for OPC UA was to define an information model, therefore the object models in Figure 5.7 and 5.8 was implemented in the address space. The result of this can be seen in Figure 5.10. The data exchange was performed by using request/response messages, and the subscription mechanism. OPC UA offers extended IT-security mechanisms, but these has not been considered in this master thesis. MDIS has chosen OPC UA as the unified architecture between the MCS and DCS (Section 2.7), and further adoption of OPC UA at lower hierarchy levels is predicted to continue within the oil & gas industry. OPC UA however is not a real-time protocol and situations where data does not arrive within a bounded time may occur. Mechanisms such as timestamps and quality markers can be used to detect link problems or missing messages. OPC UA does not seem suited for safety critical systems.

The idea behind implementing the two protocols (Modbus TCP and OPC UA) were to process the practical experience, and use it to compare the practical implementation details. The details regarding practical experience can be found in Section 5.3. The main differences are related to how difficult the protocols are to implement, the message exchange types, resource demands, and efficiency. An OPC UA SDK offers more possibilities, but the large footprint & poor efficiency represent challenges. Especially, if OPC UA is implemented at sensor-level. The communication setup of Modbus TCP is identical with the setup for OPC UA. The Modbus TCP solution is however optimized for message exchange between one client and one server. The OPC UA server already supports communication with several servers or clients. A real subsea control system tends to consist of redundant client and servers, and one client tends to communicate with several servers. Software changes must however be performed for achieving multicast communication on the embedded microcontrollers. The Modbus TCP communication is fully operational for message exchange between the two microcontrollers, but it has not been tested against other industrial equipment using Modbus TCP. Also, the interpretation of the specification, the implementation, and testing is performed by the same person. This means the same person could have misunderstood or unintentional omitted details.

As mentioned before, the initial objectives behind this project was to focus on the theoretical part by studying communication parameters such as bandwidth, payload, throughput, and send/receive optimization obtained through the benchmark comparisons. But when the two FPGA development kits were ruled out due to unreasonable cost, the project focus was shifted. Both student loan and discounts for the equipment was proposed, but in the end it was not possible to come to an agreement. The FPGA development kit intended for this project would have been delivered with all the relevant industrial Ethernet protocols and software development kits, and change of communication protocols was not supposed to be a time consuming process. When this evaluation kit was omitted I ended up using much more time on practical implementations for Modbus TCP and OPC UA. Especially, the estimated implementation time for OCP UA turned out to be quite optimistic. OPC UA represent something new in several ways compared to Modbus TCP with its services, modeling capabilities, and terminology. Even the cross-compile building of the OPC UA SDK turned out to be a time-consuming trial and error challenge. The detailed tutorials, and examples in the SDK also took some extra time to complete. However, I feel the implementation of OPC UA and Modbus TCP has been informative and provided valuable knowledge about communication, embedded micro-controller/computer systems, operating systems, and general application layer protocols.

6.2 Recommendations for Further Work

Suggested extensions related to this master thesis is given below.

- The original objective for the practical implementation was to implement the topside protocols (Modbus TCP, Ethernet/IP and OPC UA) on the same hardware and perform a detailed benchmark comparison. Due to unreasonable cost for two FPGA development kits including all the communication protocols and SDKs this was omitted. To obtain actual comparison data based on the same hardware its recommended to obtain financial funding and perform this benchmark comparison. Or, find other solutions to implement the protocols at the same hardware & network.

- As mentioned in Section 5.1.2 only the central function codes related to process data exchange between the topside controller and the embedded system in the SEM is implemented. In order to pass a Modbus conformance test and obtain a certification of the developed server and client it will be necessary to implement the function codes in class 2. The Ethernet controllers is communication and exchanging data with each other correctly, but neither the client or server is test against other industrial Modbus TCP communication equipment. Therefore, starting testing the client and server against validated Modbus TCP equipment is recommended.

- The implementation of the OPC UA serves as an example solution for a prospective OPC UA communication model in a subsea control system. If an OPC UA solution is selected, further work will include tasks such as: determining robust and reliable hardware, and select an appropriate OPC UA SDK. Other tasks will be to decide security mechanisms, develop software objects (perhaps use work from MDIS? - Section 2.7), develop object models describing specific subsea project data, and perform extensive testing.

- As described in Section 5.2 the UaExpert software was used as an OPC UA client at the host computer. This thesis demonstrates a solution for the OPC UA server, solutions for an OPC UA client located topside has not been discussed. In-depth studies for the UA client solution should be performed.

- Section 3 only discusses & compares well-known and globally approved protocols used for topside communication, and not proprietary protocols. There are proprietary protocols for this purpose that are frequently used within subsea communication. A possible extension of the project could be to include proprietary protocols such as the FMC 722 Automation protocol. This would include contacting the major subsea vendors, and perform a survey for revealing other relevant proprietary protocols. An attempt to reach out to several of the major subsea vendors was performed in this project, but it revealed little interest of sharing and publish protocol specifications.

- As mentioned in the discussion, performing a direct theoretical comparison between IWIS and CAN bus (SIIS level II) has been a challenge. An interesting extension to this section would be to implement an IWIS and CAN bus communication setup and study benchmark data, such as: response time, utilized bandwidth, and protocol conversion time. The practical implementation would also offer an possibility do a performance test which can reveal if there are any existing bottlenecks (as discussed in Section 4.3) due to the low default baud-rate of 9600 bit/sec. IWE vendors uses vendor specific protocols between the downhole tools & the downhole controller card, and for message exchange between the IWCS applications and IWE. If a benchmark comparison is performed its recommended to investigate which protocols are most used for the various networks, and implement those for obtain test results.

- Another idea for a possible extension would be to perform a project specific communication analysis. By collecting project specific data, such as the network setup (no. of client and server), type and quantity of IOs, protocol conversion time, protocol types, etc. a more thorough analysis can be performed. Send and receive optimization of messages can also be included. As mentioned in Section 5.3 Modbus TCP has several methods for requesting data from the server. The scope of this project has not covered message send or receive optimization, its therefore difficult to go into details and describe the best way of performing this. Ethernet/IP and OPC UA also defines several functions and services that can be used for data exchange.

Appendix A

OPC UA Implementation Procedure

A.1 Operating System Installation

A.1.1 Desktop Host

In order to implement OPC UA based on the SDK from Unified Automation a Windows 10 host computer must be used. The following steps must be performed:

1. Installation of Windows 10 operating system (Version 10.0.10240 or newer)

2. Installation of Visual Studio 2015 Community (Version 14.0.24720.00 Update 1), select the following add-ons during the custom installation:

   – Universal Windows App Development Tools → **Tools and Windows SDK**

   – **Windows IoT Core Project Templates** (Can also be installed directly from Visual Studio at the *Tools* tab → Extensions and Updates → Online → Search: *Windows IoT Core Project Templates*)

3. Enable developer mode in Windows 10. Setting the host machine in developer mode makes it able to install, run and test software (apps) that has not been certified by the Windows Store. It also enables the possibility of running apps from Visual Studio in debug mode. Settings → Update & Security → For developers → Choose **Developer mode**. Click Yes for the disclaimer message(s).

A.1.2 Raspberry Pi 2

For setup of the Raspberry Pi 2 model B the following hardware prerequisites must be fulfilled

- A HDMI cable and monitor
- Micro SD card reader (for the host computer)
- Ethernet cable
- 5V micro-USB supply (must deliver 1.8 A [1] or more)
- Micro SD Card. Recommended to use a Class 10 SD card with at least 8 GB

Follow these steps to perform the installation of the Windows 10 IoT Core onto the RPi 2.

1. Download the Windows 10 IoT Core Release Image onto the host desktop PC from Microsoft developer resources [2].
2. When finished downloading. Double click on the downloaded ISO-file. This will cause the ISO-file to automatically mount itself as a virtual device.
3. Double click the Windows Installer Package (.msi) and follow the installation steps, this will result in a flash image file (.ffu).
4. Download the "IoT Core Dashboard" from the link [2] for installation and configuring of Windows 10 IoT core devices. The Windows 10 IoT Core dashboard will allow you to flash the downloaded image onto your MicroSD card.
5. Insert a compliant MicroSD card into the card reader on the host machine.
6. Run the previous downloaded "IoT Core Dashboard". Click *set up a new device* → at the Device type drop down list select **Custom**. Browse to find the previous generated image file (.ffu).
7. Accept the *software license terms* and press **Install**. After the installation finishes, eject the memory card from the host computer and insert it into the MicroSD Card in the RPi. This will boot-up the Windows 10 IoT Core.

[1] https://www.raspberrypi.org/help/faqs/#powerReqs
[2] http://ms-iot.github.io/content/en-US/Downloads.htm

A.2 Initiate Connection

A basic connection test can be performed when the operating system installations in Appendix A.1 has been executed. The next step will be to connect the Raspberry Pi (RPi) to the host computer by use of an Ethernet cable. The RPi must also be connected to a HDMI supported monitor which will yield its IP-address in the *Device info* tab. Try to Ping this IP-address from the host computer in order to verify the communication setup.
At successful connection test the next steps will be:

- To remote connect and configure a Windows 10 IoT Core its recommended to use the Windows 10 built in MS PowerShell. Search for *PowerShell ISE* in the *Windows Start Menu*. Right click and select **Run as administrator**.
- Initiate a PowerShell (PS) session with the RPi.
  - Start the WinRM service to enable remote connections. Use the command:

    ```
    net start WinRM
    ```

  - Start a remote session with the RPi with the command given below. Login by enter the password. The default password of the RPi is *p@ssw0rd*. The login takes some time (approx. 30 seconds). Use the same IP-address as during the connection test.

    ```
    Enter-PSSession -ComputerName <IP-address> -Credential <
       IP-address>\Administrator
    ```

  - Its helpful to set a machine name for the RPi, e.g *mypi*

    ```
    setcomputername <machine-name>
    ```

  - Its required to reboot the device after changing the name. For a shutdown with reboot enter:

    ```
    shutdown /r /t 0
    ```

  - Create a trust relationship between the host desktop PC and the RPi. The <machine-name> was selected in the previous steps. Could also enter the IP-address instead of the machine name.

```
Set-Item WSMan:\localhost\Client\TrustedHosts -Value <
    machine-name>
```

- Re-run the command from previous steps but instead of the IP-address, use the new machine name.

```
Enter-PSSession -ComputerName <machine-name> -Credential <
    machine-name>\Administrator
```

- The firewall on the RPi will prevent the host to connect to the OPC UA server, the simplest solutions for this demonstration is therefore to deactivate the firewall. Opening the specific port for OPC UA communication is also a possibility. Commands for firewall settings:
  * Turn firewall off (deactivate)

    ```
    NetSh Advfirewall set allprofiles state off
    ```

  * Turn on firewall (activate)

    ```
    NetSh Advfirewall set allprofiles state on
    ```

  * Check status of the Windows Firewall

    ```
    NetSh Advfirewall show allprofiles
    ```

• After deactivating the firewall, continue with the next sections describing installation of the OPC UA SDK, and transferring files.

A.3 Installation & Setup of the OPC UA SDK

The OPC UA SDK contains Visual Studio 2015 C/C++ client and server projects, tutorials and a precompiled server & client. A good starting point for a simplified test of OPC UA is to use the included precompiled server for C++. This can be found in the *uaservercpp-winIoT-arm-vs2015-1.5.0-318* zipped folder. Unzip it and transfer the files in the *uaservercpp/bin* to the RPi. Further

details on how to transfer and execute files can be found in Appendix A.4. To build and transfer project files or examples from the OPC UA SDK follow the instruction steps in Section A.3.
The evaluation edition of the OPC UA SDK consists of C++ files and Visual Studio 2015 project files, all of these can be found in the zipped *uasdkcppbundle-bin-EVAL-winIoT-arm-vs2015-v1.5.0-318* folder. The evaluation edition of the SDK contains CMakeFiles which needs to be built by using the CMake application software. CMake is a tool used for building cross-platform software.

- Install the CMake GUI for Windows 10 (version 3.5.0 or newer)
- Unzip the *uasdkcppbundle-bin-EVAL-winIoT-arm-vs2015-v1.5.0-318* folder.
- Visual Studio 2015 includes compilers for 32-bit, 64-bit and ARM-based Windows operating systems that can be used to create apps. By default the Visual Studio uses either the 32 or 64-bit compiler based on the computer architecture. In case of a cross-compile to an ARM based Windows OS one must specify this by a cross-tool command prompt. Search and run the *VS2015 x86 ARM Cross Tools Command Prompt* in the *Windows Start Menu*.
- In this Cross Tool Command Prompt, use the change directory (*cd*) command to navigate to the *bin* folder of the installed CMake application and start the *cmake-gui.exe*. By using this Cross Tool Command Prompt all of the environmental variables are set according to the ARM-based system.
- When CMake starts up, browse to the unzipped evaluation edition of the UA SDK from previous steps. Normally, located inside a *sdk* folder. Set this directory for both the source code and the path for building the binaries.
- Press **Configure** and select the generator for this project in the drop down list to be **Visual Studio 14 2015 ARM**. Select the *Specify toolchain file for cross-compiling* radio button. Press **Next**, browse to the toolchain file inside the *sdk* folder - the last part of the directory path ends with: *sdk/toolchains/winIoT/toolchain-windowsIoT-arm.cmake*. Press **Finish**. CMake will then check for a valid compiler and create Makefiles.
- The final step to generate all of the Visual Studio Project files for the examples is pressing **Generate** button in CMake.

- Navigate to the *sdk* folder and double click one of the Visual Studio Solutions (.sln) in the *sdk/examples* folder.
- Open the project folder under *Examples* in the Solution Explorer in Visual Studio. Open the *Source Files* folder, click the *servermain.cpp* file.
- Change *Debug* to **Release** in the *Solution Configurations* drop down list. Navigate to the *Build* option, and press **Build Solution**. The executable project file can be found in the project *Release* folder.
- Details related to transfer, and remote commands for executing the project file see Appendix A.4.

A.4 Transfer and Run Executable Files

A.4.1 File Transfer Method

Transfer of project related files can be performed in several ways, e.g putting the Micro SD-card into the host computer and copy the files onto it, or use a file transfer protocol (ftp) client such as *WinSCP*. Its also possible to perform the operation by use of command lines in PowerShell ISE. However, the simplest solution is thought to be WinSCP. After downloading the WinSCP software, choose *FTP* as File Protocol. Host name is the <machine-name> or the IP-address of the RPi. User name is *Administrator* and the default password is *p@ssw0rd*

Create a folder at the RPi to deploy the files from the SDK, e.g call the folder *deploy* and place it in the *root* folder.

A.4.2 Mandatory Common Files

The executable project file released in Visual Studio (as described in Section A.3) and the precompiled server/client depends on common files, such as the OPC UA stack library. All of these files must always be located in the *deploy* folder at RPi. All of the common files can be found in the *sdk/bin* folder. The common files are listed below.

- The *Animation* folder
- *ClientConfig.ini* (Only used when deploying a client)
- *libeay32.dll*
- *libxml2.dll*
- *ServerConfig.ini* (Only used when deploying a server)
- *uanodesetimport.xml*
- *uastack.dll*

A.4.3 Project Dependent File

To run the precompiled-server, transfer the *uaservercpp.exe* from the *sdk/bin* folder to the deploy folder on the RPi.

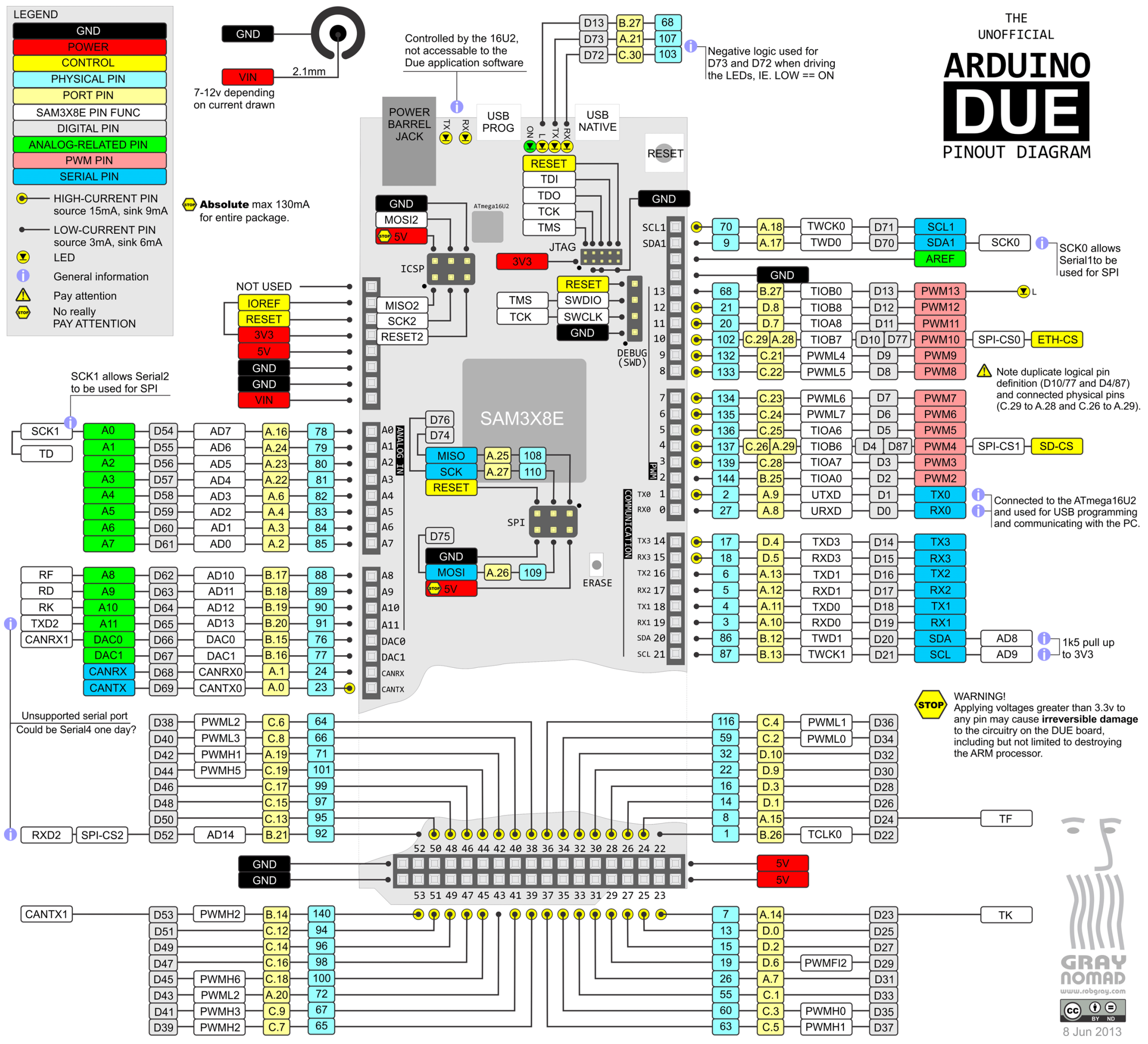
To run one of the tutorials or examples, release a executable file in Visual Studio 2015 (as described in Section A.3). Navigate to the *Release* folder of the specific tutorial/example and transfer the executable (.exe) file to the *deploy* folder on the RPi.

To start this OPC UA server/client executable file, use the PS session initiated with the RPi in Appendix A.2. The command *cd* can be used to navigate to the *deploy* folder. Start the server by writing the name of the executable project file .\<*project file name*>*.exe* into the PowerShell command. Due to an PowerShell bug its not possible to see the output from the server during execution. Its therefore recommended to use another SSH client, such as the Putty software for starting the server. More information of this bug is found in Section 5.2.3.

Appendix B

Modbus TCP Implementation Diagrams

B.1 Arduino Due Pinout Diagram



THE UNOFFICIAL
ARDUINO DUE
PINOUT DIAGRAM
LEGEND
GND
POWER
CONTROL
PHYSICAL PIN
PORT PIN
SAM3X8E PIN FUNC
DIGITAL PIN
ANALOG-RELATED PIN
PWM PIN
SERIAL PIN
HIGH-CURRENT PIN source 15mA, sink 9mA
LOW-CURRENT PIN source 3mA, sink 6mA
LED
General information
Pay attention
No really PAY ATTENTION
Absolute max 130mA for entire package.
7-12v depending on current drawn
2.1mm
Controlled by the 16U2, not accessable to the Due application software
Negative logic used for D73 and D72 when driving the LEDs, IE. LOW == ON
SCK0 allows Serial1to be used for SPI
SCK1 allows Serial2 to be used for SPI
Note duplicate logical pin definition (D10/77 and D4/87) and connected physical pins (C.29 to A.28 and C.26 to A.29).
Connected to the ATmega16U2 and used for USB programming and communicating with the PC.
1k5 pull up to 3V3
Unsupported serial port Could be Serial4 one day?
WARNING! Applying voltages greater than 3.3v to any pin may cause irreversible damage to the circuitry on the DUE board, including but not limited to destroying the ARM processor.
SAM3X8E
8 Jun 2013

Figure B.1: The Arduino Due pinout diagram [1].

B.2 Modbus Sequence Diagrams

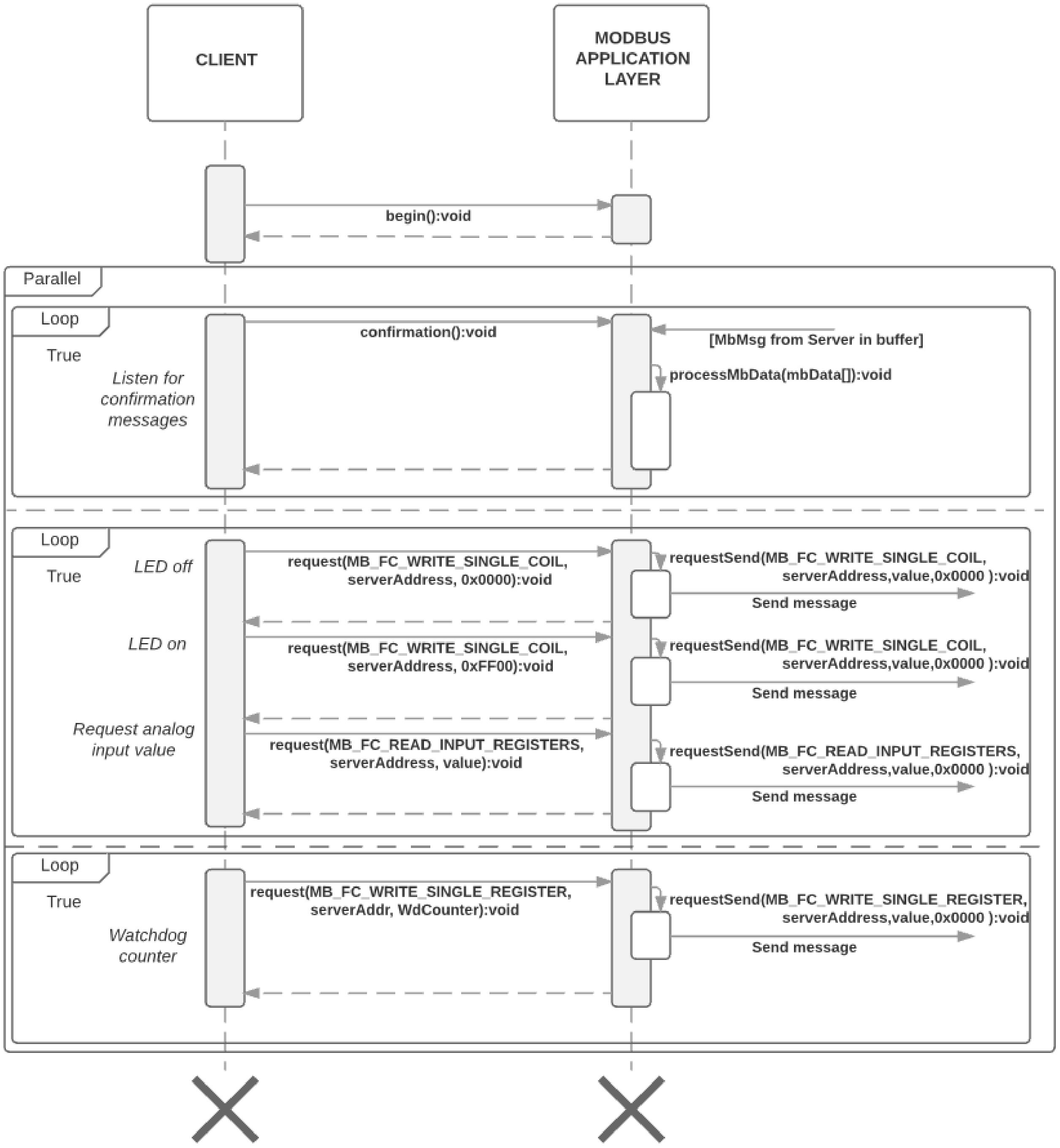

CLIENT
MODBUS APPLICATION LAYER
begin():void
Parallel
Loop
True
Listen for confirmation messages
confirmation():void
[MbMsg from Server in buffer]
processMbData(mbData[]):void
Loop
True
LED off
request(MB_FC_WRITE_SINGLE_COIL, serverAddress, 0x0000):void
requestSend(MB_FC_WRITE_SINGLE_COIL, serverAddress,value,0x0000):void
Send message
LED on
request(MB_FC_WRITE_SINGLE_COIL, serverAddress, 0xFF00):void
requestSend(MB_FC_WRITE_SINGLE_COIL, serverAddress,value,0x0000):void
Send message
Request analog input value
request(MB_FC_READ_INPUT_REGISTERS, serverAddress, value):void
requestSend(MB_FC_READ_INPUT_REGISTERS, serverAddress,value,0x0000):void
Send message
Loop
True
Watchdog counter
request(MB_FC_WRITE_SINGLE_REGISTER, serverAddr, WdCounter):void
requestSend(MB_FC_WRITE_SINGLE_REGISTER, serverAddress,value,0x0000):void
Send message

Figure B.2: Modbus TCP client sequence diagram

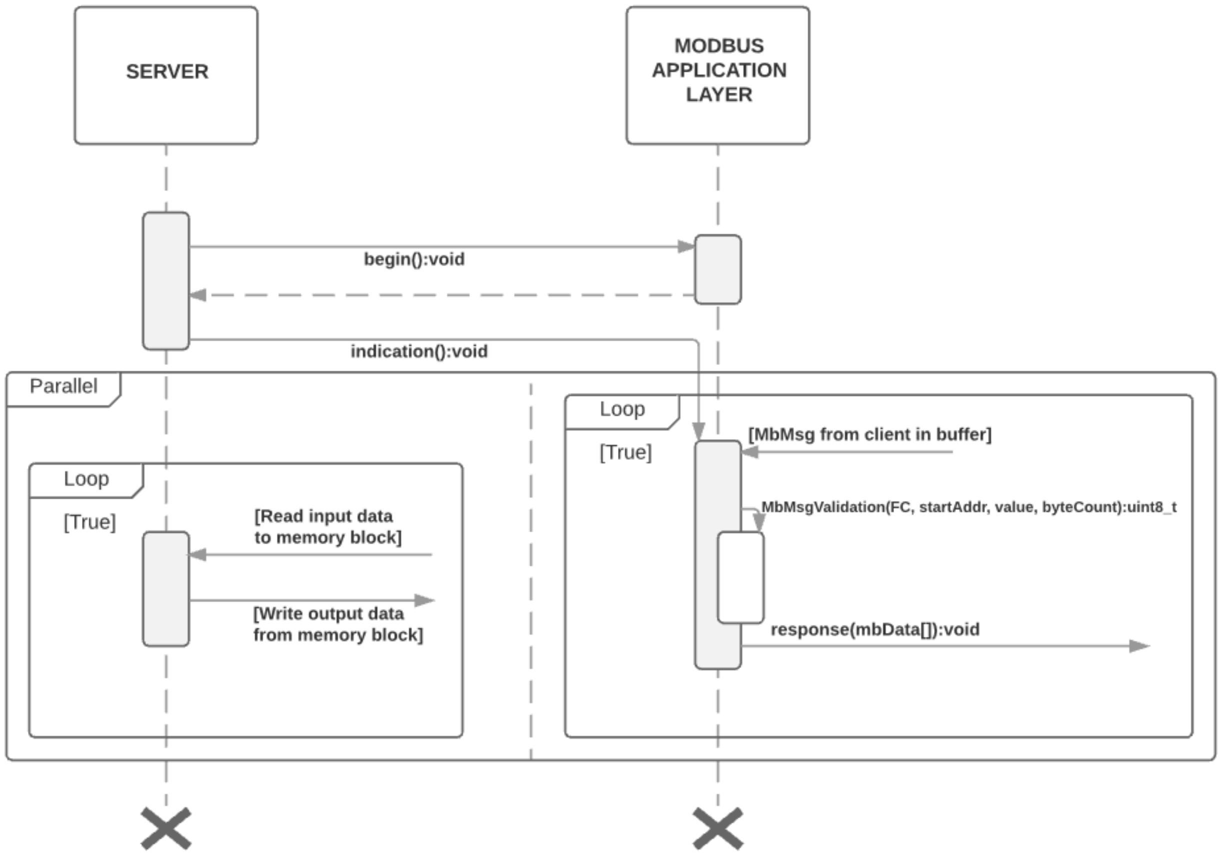

SERVER
MODBUS APPLICATION LAYER
begin():void
indication():void
Parallel
Loop
[True]
[Read input data to memory block]
[Write output data from memory block]
Loop
[True]
[MbMsg from client in buffer]
MbMsgValidation(FC, startAddr, value, byteCount):uint8_t
response(mbData[]):void

Figure B.3: Modbus TCP server sequence diagram

B.3 Class Diagrams for Modbus TCP

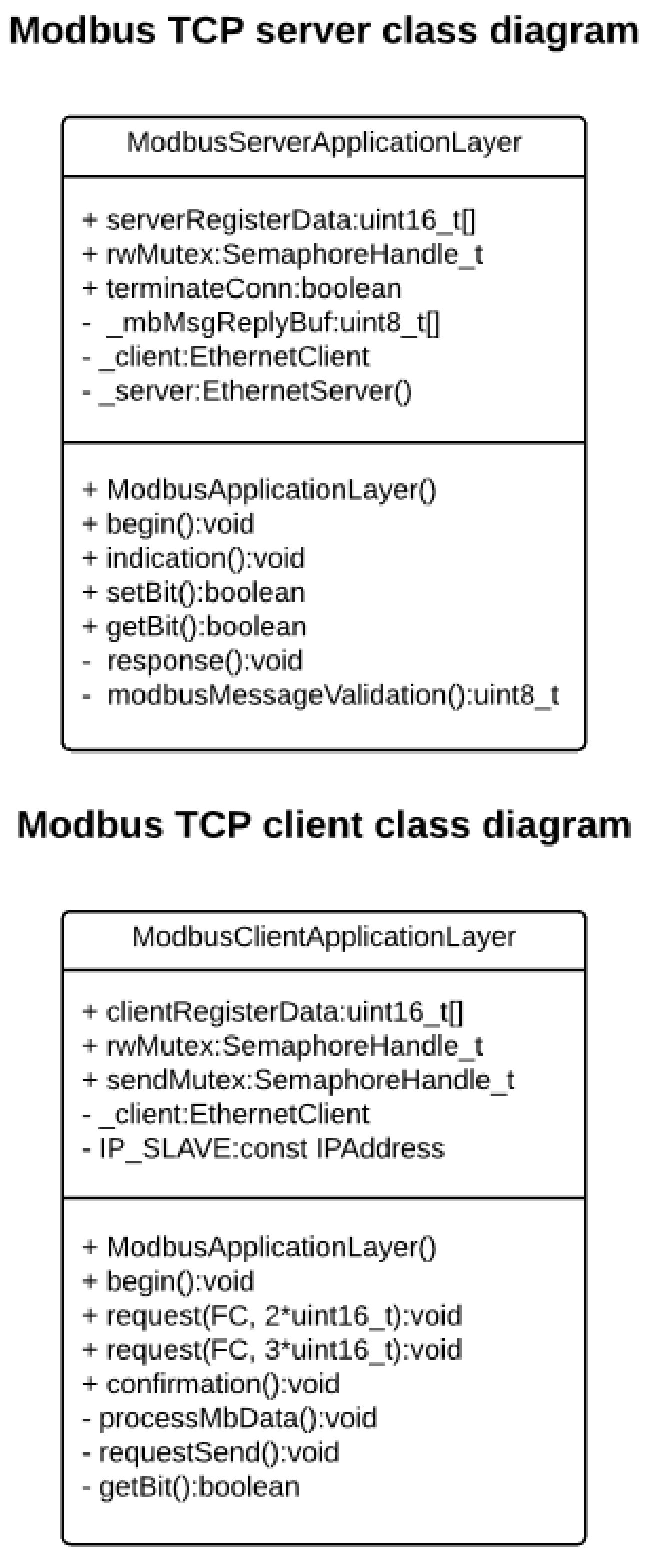

Modbus TCP server class diagram
ModbusServerApplicationLayer
+ serverRegisterData:uint16_t[]
+ rwMutex:SemaphoreHandle_t
+ terminateConn:boolean
- _mbMsgReplyBuf:uint8_t[]
- _client:EthernetClient
- _server:EthernetServer()
+ ModbusApplicationLayer()
+ begin():void
+ indication():void
+ setBit():boolean
+ getBit():boolean
- response():void
- modbusMessageValidation():uint8_t
Modbus TCP client class diagram
ModbusClientApplicationLayer
+ clientRegisterData:uint16_t[]
+ rwMutex:SemaphoreHandle_t
+ sendMutex:SemaphoreHandle_t
- _client:EthernetClient
- IP_SLAVE:const IPAddress
+ ModbusApplicationLayer()
+ begin():void
+ request(FC, 2*uint16_t):void
+ request(FC, 3*uint16_t):void
+ confirmation():void
- processMbData():void
- requestSend():void
- getBit():boolean

Figure B.4: Class diagram for the Modbus TCP client and server

Appendix C

Description of Appendix

C.1 Modbus TCP

C.1.1 Modbus TCP Server and Client Application

The Modbus TCP application layer for both the server and client microcontroller. Documentation describing the implementation can be found in Section 5.1.

C.2 OPC UA SDK

C.2.1 OPC UA SDK

A C/C++ OPC UA server/client SDK evaluation edition from Unified Automation. An evaluation edition is limited to run for only one hour at the time (it needs a simple restart after this). The appendix contains evaluation editions for Raspberry Pi 1 & 2 and the Beagle Bone Black. For this thesis the OPC UA SDK bundle *uasdkcppbundle-bin-EVAL-winIoT-arm-vs2015-v1.5.0-318* is most relevant. The SDK must be unzipped, and built using CMake. Section A.3 describes the procedure.

C.2.2 OPC UA Project File

The Visual Studio 2015 project files for implementation of the OPC UA server as described with the object models illustrated in Figure 5.7 and 5.8.

References

[1] The Unoffical Arduino Due Pinout Diagram. http://www.robgray.com/temp/Due-pinout-A4.png.

[2] Acromag, Inc. (2005). Introduction to Modbus TCP/IP. http://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf.

[3] Adriaansen, L. (2004). *Subsea Control and Data Acquistion*. Professional Engineering Publishing, Bury St Edmunds.

[4] American Petroleum Institute (2014). Standard for Subsea Production Control Systems. http://www.techstreet.com/api/products/1877245.

[5] Arduino (2016a). https://www.arduino.cc/en/Main/ArduinoBoardDue.

[6] Arduino (2016b). https://www.arduino.cc/en/Main/ArduinoEthernetShield.

[7] Bai, Y. (2012). *Subsea Engineering Handbook*. Elsevier Science, Amsterdam.

[8] CAN in Automation (2011). Cia 301 - version 4.2.0 (public). http://www.can-cia.org/standardization/specifications/.

[9] Caro, D. (2009). *Automation Network Selection: A References Manual*. International Society of Automation, 2nd edition.

[10] Cavalieri, S. and Chiacchio, F. (2013). Analysis of OPC UA performances. http://www.sciencedirect.com/science/article/pii/S0920548913000640.

[11] Cavalieri, S., Cutuli, G., and Monteleone, S. (2010). Evaluating Impact of Security on OPC UA Performance. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5514495&tag=1.

[12] Cisco Systems, I. (2003). Industrial ethernet: A control engineers's guide. http://www.cisco.com/c/dam/en/us/products/collateral/switches/catalyst-2950-series-switches/prod_white_paper0900aecd8013313e.pdf.

[13] Corneliussen, S. and Lundanes, E. (2009). IWIS and SIIS in Subsea Control Systems: Experience from Implementation. http://mac.sagepub.com/content/42/4/113.full.pdf+html.

[14] Decotignie, J.-D. (2005). Ethernet-based real-time and industrial communications. http://isa.uniovi.es/~sirgo/doctorado/ethernet_en_tiempo_real.pdf.

[15] Farsi, M. and Barbosa, M. B. M. (2000). *CANopen implementation: applications to industrial networks*. Research Studies Press LTD., Baldock, Hertfordshire, England.

[16] Felser, M. (2005). Real-Time Ethernet - Industry Prospective. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1435742.

[17] Fojcik, M. and Folkert, K. (2012). *Computer Networks: 19th International Conference, CN 2012, Szczyrk, Poland, June 19-23, 2012 Proceedings*, chapter Introduction to OPC UA Performance, pages 261–270. Springer Berlin Heidelberg, Berlin, Heidelberg.

[18] Hunkar, P. (2015). OPC UA and MDIS Complete Successful Interoperability Test. https://opcfoundation.org/opc-connect/wp-content/uploads/2015/06/OPCUA_MDIS_IOP.pdf.

[19] IEC (2014). Industrial communication networks profiles. IEC 61784 4:2014, International Electrotechnical Commission, Geneva, Switzerland.

[20] International Organization for Standardization (ISO) (2006). ISO 13628: Petroleum and natural gas industries - Design and operation of subsea production systems - Part 6: Subsea production control systems. https://www.standard.no/en/webshop/productcatalog/productpresentation/?ProductID=171400.

[21] Koeptz, H. (1991). Event-triggered versus time-triggered real-time systems. http://itech.fgcu.edu/faculty/zalewski/CEN3213/pdf/KopetzTTvsET-1991.pdf.

[22] Mahnke, W., Leitner, S.-H., and Damm, M. (2009). *OPC Unified Architecture*. Springer, Ladenburg, Germany, 1st edition.

[23] Marshall, P. S. and Rinaldi, J. S. (2004). *Industrial Ethernet - How to Plan, Install, and Maintain TCP/IP Ethernet Networks: The Basic Reference Guide for Automation and Process Control Engineers*. ISA - The Instrumentation, Systems and Automation Society, United States of America, 2nd edition.

[24] Modbus Organization, Inc. (2006). Modbus Messaging on TCP/IP Implementation Guide v1.0b. http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.

[25] Modbus Organization, Inc. (2012). Modbus Application Protocol Specification V1.1b3. http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[26] ODVA (2006). The Common Industrial Protocol (CIP) and the Family of CIP Networks. https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00123R0_Common-Industrial-Protocol-and-Family-of-CIP-Netw.pdf.

[27] ODVA (2013). Ethernet/IP - Quick start for vendors handbook. https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00213R0_EtherNetIP_Developers_Guide.pdf.

[28] OPC Foundation (2015). OPC Unified Architecture Specification, Parts 1-13. https://opcfoundation.org/developer-tools/specifications-unified-architecture/.

[29] OTM Consulting Ltd (2006). IWIS: background, terms of reference. http://www.iwis-jip.com/index.asp.

[30] OTM Consulting Ltd (2011). IWIS Recommended Practice. http://www.iwis-jip.com/docs/IWIS-RP-A2-Master_Apr_2011.pdf.

[31] OTM Consulting Ltd (2015). SIIS Recommended Practice (V008 Master). http://www.siis-jip.com/index.php/documents/2015-22-10-siis-rp-v008-master-public.

[32] Park, J., Mackay, S., and Wright, E. (2003). *Practical Data Communications for Instrumentation and Control*. Elsevier, Jordan Hill - Oxford.

[33] Petroleum Safety Authority Norway (2015). The management regulations. http://www.psa.no/management/category401.html.

[34] Raspberry Pi Foundation (2015). https://www.raspberrypi.org/products/.

[35] Real Time Automation. http://www.rtaautomation.com/technologies/modbus-tcpip/.

[36] Robert, J., Georges, J.-P., Rondeau, E., and Divoux, T. (2012). Minimum cycle time analysis of ethernet-based real-time protocols. http://univagora.ro/jour/index.php/ijccc/article/viewFile/1372/357.

[37] Sande, O., Fojcik, M., and Cupek, R. (2010). *Computer Networks: 17th Conference, CN 2010, Poland, June 15-19*, chapter OPC UA Based Solutions for Integrated Operations, pages 76–83. Springer Berlin Heidelberg, Berlin, Heidelberg.

[38] Saul, D. (2006). Subsea instrumentation interface standardization in the offshore oil and gas industry. http://www.can-cia.org/fileadmin/resources/documents/proceedings/2006_saul.pdf.

[39] Scheler, F. and Schröder-Preikschat, W. (2006). Time-Triggered vs. Event-Triggered: A matter of configuration? http://www4.cs.fau.de/Publications/2006/scheler_06_mmb-nfp.pdf.

[40] Unified Automation. https://www.unified-automation.com/.

[41] Vector Informatik GmbH. http://canopen-solutions.com/canopen_fundamentals_en.html.

[42] Welander, P. (2014). Integrating automation offshore. http://www.oedigital.com/component/k2/item/6411-integrating-automation-offshore.

[43] Wilamowski, B. M. and Irwin, J. D. (2011). *Industrial Communication Systems*. CRC Press, Boca Raton, FL, 2nd edition.

[44] WIZnet Co. Inc. (2016). `http://www.wiznet.co.kr/product-item/w5100/`.

[45] Zeltwanger, H. (2009). Seminar presentation - Future of CANopen (CAN bus subsea photo). `https://www.tekes.fi/globalassets/global/ohjelmat-ja-palvelut/ohjelmat/ubicom/aineistot/tilaisuuksien-ja-seminaarien-materiaalit-1/20090611_teollisuusvaylat/seminaari_20090611-canopen-holgerzeltwanger-can-cia.pdf`.

[46] Zurawski, R. (2015). *Industrial Communication Technology Handbook*. CRC Press, Boca Raton, FL, 2nd edition.