Aalto University
School of Electrical Engineering

Joona Elovaara

Aggregating OPC UA Server for Remote Access to Agricultural Work Machines

Espoo 30.6.2015

Supervisor: Prof. Arto Visala
Instructors: D.Sc. Ilkka Seilonen, D.Sc Timo Oksanen

Aalto-yliopisto

AALTO-YLIOPISTO TEKNIIKAN KORKEAKOULUT PL 12100, 00076 Aalto http://www.aalto.fi	DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Joona Elovaara

Työn nimi: Aggregoivan OPC UA palvelimen käyttö etäyhteydessä maatalouden työkoneisiin

Korkeakoulu: Sähkötekniikan korkeakoulu

Laitos: Sähkötekniikan ja automaation laitos

Professuuri: Autonomiset järjestelmät	Koodi: Aut-84

Työn valvoja: Prof. Arto Visala

Työn ohjaajat: TkT Ilkka Seilonen, TkT Timo Oksanen

Maataloustyökoneet ovat kehittyneet nopeasti viime vuosikymmeninä ja elektronisten ohjainten ja anturien rooli niiden toiminnassa on lisääntynyt. Samaan aikaan, ja osittain tästä johtuen, myös maanviljelyyn liittyvät tehokkuusvaatimukset ja erilaiset säännökset ovat lisääntyneet. Nykyisin tietojärjestelmät, jotka auttavat maanviljelijää erilaisissa maatilan hoitamiseen liittyvissä asioissa, ovat yleistyneet. Useat työkoneet käyttävät ISO 11783 -tiedonsiirtoprotokollaa, jonka avulla näistä työkoneista voidaan lukea paljon niiden toimintaan ja sisäiseen tilaan liittyvää dataa. Tätä dataa ei kuitenkaan tällä hetkellä kerätä eikä hyödynnetä niin hyvin kuin olisi mahdollista. Tämä diplomityö arvioi aggregoivia OPC UA palvelimia mahdollisena teknologiana, jolla useista työkoneista saatavaa dataa voitaisiin keskitetysti lukea ja tallentaa.

Tämä diplomityö perustuu aiempaan diplomityöhön, jonka tuloksena syntyi OPC UA -tietomalli ISO 11783 -tiedonsiirtoprotokollaa hyödyntäville maataloustyökoneille. Tässä työssä tehdään ensin kirjallisuuskatsaus maatilanhallinnan nykytilanteeseen ja suunniteltuun lähitulevaisuuteen, sekä siihen, kuinka ja missä työkoneista saatavaa dataa voitaisiin hyödyntää. Tämän katsauksen ja edellisen työn perusteella määriteltiin vaatimukset aggregoivalle OPC UA palvelimelle, joka keskittää useasta OPC UA -palvelinta käyttävästä työkoneesta saatavan tiedon. Palvelimesta suunniteltiin ja kehitettiin prototyyppi, joka pystyy automaattisesti muuntamaan tiedot useasta ISO 11783 -protokollaa käyttävästä työkoneesta yhtenäiseksi näkymäksi hyödyntämällä edellisessä työssä kehitettyä tietomallia vasten kirjoitettuja sääntöjä.

Aggregoivat OPC UA –palvelimet todettiin kelvolliseksi teknologiaksi keskitettyyn tiedonkeruuseen ISO 11783 –protokollaa käyttävistä työkoneista. Kaikkeen oleelliseen dataan, joka on saatavilla OPC UA:ta hyödyntävistä työkoneista, on pääsy myös aggregoivan palvelimen läpi. Aggregoivan palvelimen käyttämiä muunnossääntöjä voidaan laajentaa keskittämään tietoa myös muista OPC UA:ta hyödyntävistä laitteista. Aggregoivien palvelinten hyödyllisyys tulee lisääntymään, jos OPC UA tullaan ottamaan yleisemmin käyttöön ja jos sitä hyödyntävien laitteiden määrä maataloudessa lisääntyy.

Päivämäärä: 30.6.2015	Kieli: Englanti	Sivumäärä: 7+68

Avainsanat: OPC UA, aggregoivat palvelimet, datan kartoitus, ISOBUS, FMIS

Aalto-yliopisto

AALTO UNIVERSITY SCHOOLS OF TECHNOLOGY PO Box 12100, FI-00076 AALTO http://www.aalto.fi	ABSTRACT OF THE MASTER'S THESIS

Author: Joona Elovaara

Title: Aggregating OPC UA Server for Remote Access to Agricultural Work Machines

School: School of Electrical Engineering

Department: Department of Electrical Engineering and Automation

Professorship: Autonomous Systems | Code: Aut-84

Supervisor: Prof. Arto Visala

Instructor(s): D.Sc Ilkka Seilonen, D.Sc Timo Oksanen

Agricultural machinery has improved rapidly during the last decades and the role of electronic controllers and sensors has increased. Simultaneously and partially because of this, the efficiency requirements concerning farming have grown more stringent. Information systems to help in farm management are already commonplace. Many work machines utilize the ISO 11783 communication protocol, through which a lot of data from the operation and internal state of a machine could be retrieved. However, this available data from agricultural work machines is not fully utilized. This thesis evaluates OPC UA aggregating servers as a centralized means to read and store the data available from multiple work machines.

This thesis builds on an earlier thesis where an OPC UA information model was developed for agricultural work machines utilizing the ISO 11783 communication protocol. First, a literature review was done to get a general view of the current state and the intended future of farm management systems, and to see how and where the data from the work machines can be utilized. Based on this and the previous work, the requirements for an aggregating server concentrating the information available from work machines running OPC UA servers were defined. A prototype aggregating server that is able to automatically transform the information from multiple ISO 11783 work machines to a unified view using a set of mapping rules was designed and implemented.

OPC UA aggregating servers were found to be viable technology for the centralized data monitoring and collection of ISO 11783 work machines. All relevant data exposed by the work machine OPC UA servers can also be accessed through the aggregating server. The mapping engine implemented on the aggregating server prototype can be extended to automatically map the information from other devices exposing themselves through OPC UA as well. The usefulness of an aggregating server increases if in the future OPC UA will be more commonly adopted and utilized by other agricultural equipment as well.

Date: 30.6.2015	Language: English	Number of pages: 7+68

Keywords: OPC UA, aggregating servers, data mapping, ISOBUS, FMIS

Preface

I would like to thank my instructors D.Sc. Ilkka Seilonen and D.Sc. Timo Oksanen for their diligent guidance and help during this thesis and for putting up even with the more stupid questions.

I would also like to thank Sami Pietikäinen from Wapice for his dedicated help with technical issues during the earlier part of working with this thesis.

Finally I would like to thank my family, especially my mother, whose unwavering encouragement helped keep me going during the whole course of my studies.

Otaniemi 30.6.2015

Joona Elovaara

Contents

Abbreviations

CAN	Controller Area Network
COM	Component Object Model
DCOM	Distributed Component Object Model
DDOP	Device Description Object Pool
EC	Electrical Conductivity
ECU	Electronic Control Unit
FMIS	Farm Management Information System
GNSS	Global Navigation Satellite System
GPR	Ground Penetrating Radar
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
GUID	Globally Unique Identifier
HMI	Human Machine Interface
ISO	International Organization for Standardization
LHS	Left Hand Side
MIS	Management Information System
OLE	Object Linking and Embedding
OPC	Open Platform Communications
PDU	Protocol Data Unit
PGN	Parameter Group Number
RFID	Radio Frequency IDentification
RHS	Right Hand Side
SAE	Society of Automotive Engineers
SAR	Synthetic Aperture Radar
SCADA	Supervisory Control And Data Acquisition
SDK	Software Development Kit
UA	Unified Architecture
URI	Uniform Resource Identifier
WLAN	Wireless Local Area Network
WRM	Wapice Remote Management
XML	Extensible Markup Language

1 Introduction

1.1 Background

In the recent years, agricultural technology has seen rapid improvement. This has been accelerated to a large extent by improvements in different technological fields like automated processes, real-time guidance, and vision systems, all of which are related to modern agriculture (Ronkainen, 2014). Also, the advent of precision agriculture and variable rate application has further boosted the technological development of agricultural work machines (Ronkainen, 2014).

In addition to and partly because of this technological progress in the actual work machines, the requirements of managing a farm have increased and grown more complex. A farmer needs to make decisions on how and when and in which order to execute the various farming operations, he must comply with different increasing environmental regulations to avoid sanctions and to receive subsidies, he must manage finances and his work machine fleet. All this can be an overwhelming task for the farmer, which is why many farmers use tools designed specifically to assist in farm management. These tools are called farm management information systems (FMIS).

FMISs are systems designed to assist a farmer in decision making related to the different aspects of farm management. Sørensen et al. (2010a) define an FMIS as a "system for the collecting, processing, storing and disseminating of data in the form of information needed to carry out the operations functions of the farm". In addition to assisting in the actual farming operation, an FMIS is also expected to offer support in other farm management tasks such as managing finances and complying with regulations.

Due to the rapid improvement in agricultural technology and the advent of concepts such as precision agriculture and fleet management, some new requirements or features for future FMISs have been proposed for example by Sørensen et al. (2010a, 2011), Murakami et al. (2007) and Nikkilä et al. (2010). Since most tractors and implements nowadays are electronically controlled and contain many sensors and actuators, a lot of data could be collected from them that could, for example, provide information about machine maintenance and malfunctions, or about what was actually done in the field by each machine during farming operations. This information could in turn be used to improve precision agriculture and fleet management practices. In other industries like process automation, collecting data from the machines has been common practice for years. However, in agriculture, the collection, representation and utilization of this data is still relatively new and requires further research.

A lot of modern agricultural tractors and implements are electronically controlled devices that use the ISO 11783 communication protocol for internal data transfer. ISO 11783 (often abbreviated as ISOBUS) is a standardized communication protocol for agricultural equipment based on the SAE J1939 standard. The standard specifies a serial

data network for control and communications of agricultural or forestry tractors and the implements that are attached to them for different tasks, standardizing the data transfer between the components that form such devices, like sensors, actuators and controllers (ISOBUS Specifications part 1, 2006). These tractors and implements communicate a lot of data that could be utilized to improve the various farming operations, but this data is rarely collected or used. This is partly because of the lack of a standardized technology for on-line data collection from work machines.

One possible technology to implement this data collection with is OPC UA (OPC Unified Architecture), sometimes referred to simply as UA. OPC (formerly Object Linking and Embedding for Process Control, currently Open Platform Communications) is a standard specification defining communication of on-line data from various devices from different manufacturers. It is maintained by the OPC Foundation and it has gained wide acceptance in plant level data access and monitoring applications in many industries. However, the original OPC standard, now referred to as OPC Classic, has been in use for a long time already and has its drawbacks. In 2008, the OPC Foundation released a new standard called OPC UA, which aims to tackle the shortcomings of OPC Classic while introducing new features and broadening the scope of OPC. OPC UA has been under active development during the last few years, and while it is generally accepted that its integration capabilities are unrivaled by those of OPC Classic, its adaptation in different industries has not yet reflected this view.

The two main components of an OPC UA system are servers and clients. Servers are implemented on some underlying system or device, for example an ISOBUS work machine, to expose data from it in a standardized manner. Clients are used to connect to servers and access this information. The basic principle is that the servers always expose the data in a standardized way, so any client can read the data without prior knowledge of the server. This client-server architecture is highly scalable. For example, a single server can be used to connect to multiple other servers and to represent their information in a unified manner. This way a client connecting to the server can access the data of multiple servers from a single source. This kind of server configuration is called an aggregating server.

The Crop, Livestock And Forest Integrated Solution (CLAFIS) is an EU project targeted at increasing the overall effectiveness of farming by integrating data collection from different kinds of agricultural and forestry equipment and by the better utilization of this data. As a part of this project, in a previous thesis a prototype OPC UA server was designed and implemented for ISOBUS work machines by Piirainen (2014). This server enables data access through OPC UA to tractors and implements that are connected to an ISOBUS communication network. The server exposes the data from these machines in an understandable and well organized way. Implements can be added to a tractor running the server in a plug-and-play manner, i.e. when an implement is connected to the tractor, it is automatically configured to the UA server. This thesis is also a part of the CLAFIS project.

1.2 Objectives

The research problem of this thesis is to evaluate an aggregating OPC UA server as a technology for centralized remote access to agricultural work machines. The requirements for such a system are not yet fully known, nor are there existing solutions for how such a system should be designed. This thesis aims to find out possible solutions to these problems and to evaluate them by implementing and testing a prototype. This thesis focuses especially on the automatic configuration of different agricultural work machines to the aggregating server.

The scope of this study is limited by the previous work done within the same project since this thesis is premised on an information model created earlier for agricultural work machines utilizing ISOBUS. The assumptions to limit the problem are:

- All work machines utilize the ISOBUS communication protocol.
- All work machines have a pre-existing OPC UA server exposing the data from them.
- The OPC UA servers of these machines use the information model created for ISOBUS machines by Piirainen (2014).

The objectives of this thesis with regards to the research problem are:

- Define the requirements for an OPC UA aggregating server to be used in the agricultural domain. These requirements are derived from the functional needs of an aggregating server for this specific application domain, as well as from the functionality of the underlying servers connected to the aggregating server.
- Design the aggregating server. This design must account for the defined requirements for an aggregating server such as the required services, as well as the semi-automatic mapping of different address spaces from the underlying UA servers to the aggregating servers address space.
- Evaluating the functionality of this design by implementing a prototype aggregating server and testing its operation with example use cases.

It should be noted that the evaluation of the aggregating server design is also an evaluation of the previously created UA for ISOBUS design, or rather the current state of the part of the CLAFIS project that so far consists of these two theses. The focus of this thesis is on address space mapping, so to evaluate it, around 20 different dummy implements have to be instantiated to the address space of the ISOBUS UA server prototype and then mapped to the address space of the aggregating server.

1.3 Research Methods

The research framework for this thesis is design science. Hevner et al. (2004) define design science as a research framework for creating and evaluating information technology artifacts intended to solve identified problems. How design science is applied to the research process depends on the author as well as the application domain it is applied to (Peffers et al., 2006).

In this thesis, the artifacts in question are the design and prototype implementation of the aggregating server, and the design science research process is divided into four phases: requirements definition, design, implementation and evaluation. Since there are not yet well defined requirements for this type of system in this domain, they must be improved. Based on these requirements, a design for the solution is created. This thesis is not intended to bring new information to the implementation phase, since only a proof-of-concept prototype is created based on the design, which in itself is a work in progress. The design is evaluated by testing the prototype. The prototype must be able to automatically map the address space from the devices used in testing to its own address space, while implementing the desired functionality for the data exposed by the underlying server.

Before presenting the results of the design science process, a literature study is conducted to the current state of technology in agricultural work machines and FMISs. The capabilities of OPC UA and aggregating servers are also explored thoroughly. The workings of ISOBUS are studied sufficiently to give a clear picture of the design and implementation of the prototype ISOBUS UA server.

1.4 Structure of the work

First, in chapter 2, to get a general view of the application domain, FMISs and the ISOBUS communication protocol are briefly introduced, as well as some of the technical requirements of modern farm management. After this, in chapter 3, OPC UA, being the main technology of this thesis, is presented as thoroughly as needed. This chapter also introduces the previously created OPC UA for ISOBUS information model.

Chapter 4 begins the practical part of this thesis, first defining the requirements for the aggregating server design. Chapter 5 presents the actual design for the aggregating server based on the requirements definition. Chapter 6 shows the test setup and results for the prototype server and the mapping algorithm. Chapter 7 concludes the thesis by summarizing what was done and learned, and by proposing possible future improvements for the design.

2 Remote access to agricultural work machines

2.1 Farm management

Managing a farm is a continuous and complicated process that needs to consider a lot of different factors. In the end, farm management comes down to the farmer making and implementing decisions to the best of his ability to maximize the profit from his farm. These decisions have to account for a lot more than the actual farming process. A single decision may be affected by all of the different aspects of farm management.

The complex nature of farm management is introduced well by Bliss (2015). First of all, the farmer must consider the situation of his farm and which kind of farming it is most suited for. For example, the nutrient content, humidity and other conditions of his soil may be more favorable to some crops than others. Also there may be some regional hostile factors, like weeds, diseases and pests that may affect this decision. The farmer must also consider the economic situation. In deciding the crops to cultivate, he must account for the current seed price as well as the supply and demand of said crops. Also, depending on where the farm is located, some crops might be more profitable than others due to external factors like government regulations and subsidies. When the preliminary decisions are made, there are also a lot of decisions involved in the implementation of the farming. These include choosing what kind of machinery to invest in, what kind of irrigation systems are needed, designing drainage etc. Naturally, the economic aspect is closely bound to these decisions as well.

It should also be noted that farm management is very farm and farmer dependent. Naturally, the geographical location of the farm greatly affects what kind of farming it is suitable for. Even in the same country, different farms focus on different crops, which can cause a lot of different requirements for farm management. However, the most significant factor in farm management is the farmer himself. He makes the decisions, either based on scientific research, past knowledge or a hunch. Farmers often have a lot of tacit knowledge and deeply rooted assumptions and approaches. One way of doing things is not necessarily better or worse than another. Thus there exists no single standardized and optimal way of running a farm.

2.2 Farm management information systems

As more and more sophisticated agricultural technology and practices have become common in developed countries, the effective management of a farm has in many cases become too much for the farmer to handle without some management information system (MIS). A management information system is a common concept in many organizations regardless of the field of industry. The general purpose of an MIS is to analyze and facilitate activities in an organization, and to aid or automate human decision making (O'brien, 1999).

A management information system designed specifically for farming operations is called a farm management information system. Sørensen et al. (2010a) define an FMIS

as a “system for the collecting, processing, storing and disseminating of data in the form of information needed to carry out the operations functions of the farm”. On top of the actual farming operations, the FMIS should assist the farmer to survive in a world of increasing complexity by aiding in complying with the different restrictions, regulations, requirements and guidelines related to production as well as environmental factors, product quality, growing conditions and subsidies.

The planning of farming operations based on the collected data can be automated to some degree but usually the farmer is either needed or wants to participate in farm operation planning. Thus, one of the most important roles of an FMIS is to provide information for the farmer in a relevant and understandable way to assist the farmer in decision making. Sørensen (2010a) divides the functional requirements of an FMIS into four components: internal data collection, external information collection, plan generation and report generation. Murakami (2007) listed as some of the most important functional requirements a design aimed at the specific needs of the farmers, simple and easy to use interface, automated methods for data processing, a user controlled interface for accessing processing and analysis functions, integration of expert knowledge and user preferences, and enhanced integration and interoperability in general. Regardless of where you look for the requirements for an FMIS, the most often mentioned seem to be the collection and processing of data and a user friendly interface allowing access to and control of the different parts of the FMIS. The requirements and implementations of FMIS functions are examined in a bit more detail later in this section.

A lot of studies have been conducted on the design and requirements of FMISs. From these studies, some of the more common problems involving FMIS implementation emerge. Nikkilä (2010) and Sørensen (2010b) suggest that one of the most common flaws in FMISs and one of the most common obstacles for their adoption are confusing and superfluous interfaces that have not been designed with the end user in mind. The design of an FMIS can be a daunting task, and while trying to figure out how to implement all of the practical requirements, it can be easy to forget that the end user is often a single farmer, who might not be very adept in the use of any software, let alone in adopting an extensive system like an FMIS. Also, farming is not a very generalized industry, in the sense that a single farmer might have a lot of tacit knowledge or preferences which can be impossible to predict from a purely engineering point of view. Because of this, FMISs should also be as customizable and flexible as possible to give room for the farmer’s preferences, further complicating the design task.

Another difficult issue is related to data acquisition and processing. FMISs could benefit greatly from collected data from sources such as the process and internal data from the work machines, sensors measuring different qualities of the soil and from external services, for example from a weather service. As of yet, there is still no accepted standard way of extracting on-line information from the work machines. Many technologies have been suggested, each having their benefits and drawbacks. There can be a lot of different kinds of machines in the farm, and on top of this, data from multiple sites, which are

not necessary interrelated, may be collected to the same system (Sørensen, 2010a). Since a lot of these devices, sensors and services can use different standards and data formats, a lot of interoperability issues may arise from data transformations alone (Nikkilä, 2010).

However, an even bigger issue than collecting and concentrating lots of different kinds of data is analyzing and processing this data to meaningful information that can actually assist the farmer in managing his farm. Sørensen (2010b) points out, that especially in precision agriculture, the management of information and decision making based on that analysis is a more fundamental issue than acquiring the data. Sørensen (2010a) notes, that this analyzing and decision-making process is for the most parts done manually, which is very labor intensive and time consuming. For this reason, large portions of information may often not be utilized at all.

Sørensen (2010a) divided their FMIS model into functional components and suggested implementations for them. These are shown in Table 1.

Table 1. Functional components for a possible FMIS, taken directly from (Sørensen, 2010a)

Components	**Possible Implementations**
Farm activity monitoring	Sensor readings from process activities (e.g. fuel consumption, yields measurements, RFID, GPS)
Data acquisition	Capturing of the sensor data and possibly presentation to the user (e.g. tractor terminal, mobile terminal, individual implement displays)
Data transfer	Transfer of the acquired data from the point of creation to some database or processing unit (e.g. GSM, GPRS, WLAN wireless technologies)
Data processing	Processing unit aggregating and/or deriving targeted indicators (e.g. central database, web-server, application logic)
Internal repository	Database holding information on the "operations history" of the farm (e.g. local database on the farmer pc or central database, such as the personalized web-database Danish field database)
Search internal information	Locate specific information in the internal repository (e.g. specific search application logic)
Documentation generation	Derivation of indicators to evaluate compliance with norms, standards, etc. (e.g. special designed tools for specifying realized application rates, realized yields, etc.)
Extract to audit	Extraction of specific information for auditing (e.g. using specialized tools to extract the required and contextualized information)
Automated validation	Comparison between documentation and planned activities (e.g. specific tool for automated comparison)
Search external information	Locate specific information in the external distributed repository (e.g. specific search application logic for locating adverse sorts of guidelines for farming activities)

Information filtration	Contextualization and specification of the needed information for planning purposes (e.g. application software for sorting and transforming data/information into the right formats, etc.)
Operations plan generation	Decision making and plan generation for the farming processes and operations (e.g. specialized planning software modules listing predicted application rates, machinery input, labor input, etc.)
Plan repository	Repository holding and listing the generated plans at specific times (e.g. a dedicated database)
Plan execution	Actual execution of the planned activities (e.g. invoking different control system – for example downloading task files to the tractor controller for subsequent execution and control by the implement control unit (ECU))

In another paper, Sørensen (2011) presented a comprehensive list of data requirements and descriptions for a fertilizing use case. The data requirement table is unnecessarily large for the scope of this thesis, so a trimmed version of it is presented in appendix 1. Table 1 and appendix 1 both highlight the fact that data acquisition from the work machines is a relevant part of the future of farm management. Two farming concepts that would most benefit from the fully implemented work machine data acquisition are precision farming and fleet management. These are briefly introduced in the next sections.

2.3 Precision farming

Mondal et al. (2004) define precision farming as a scientific approach to improve agricultural management by the application of information technology and satellite positioning to identify, analyze and manage the spatial and temporal variability of agronomic parameters within a field by the timely application of only the required amount of a substance input to maximize profitability and sustainability while minimizing the environmental impact. The concept of precision farming emerged already in the 1980's. However, it has developed significantly as the technologies it is dependent on have developed. In a paper published in 2007, Mondal & Tewari group the technologies that currently contribute to precision farming. These are positioning systems, yield mapping, remote sensing, soil and crop sensing, variable rate technology and information transmission in farming.

A positioning system is needed in precision farming to track the movements and position of the tractor to detect where in a field certain activities should be done or have been done. A positioning system is easy to implement since satellite positioning technologies like the GPS (Global Positioning System) are nowadays commonplace.

Yield mapping measure either the volume or mass rate of the harvested crop. Yield is one of the most significant indicators of successful precision farming. Collected data of the spatial variabilities of one year's yield size can be used in evaluating the current plan and in improving next year's plan.

Remote sensing consists of technologies intended to gather electromagnetic data from the surface or subsurface of the field. Examples of these technologies include Synthetic Aperture Radar (SAR) images and a Ground Penetrating Radar (GPR). These can be used for example to map weeds against bare soil, or to map the humidity of the soil beneath the surface (Mondal & Tewari, 2007).

Soil and crop sensing measures different properties of the soil and crop using direct contact or proximate remote sensing technology to replace measurements that would otherwise need to be done from samples in a laboratory (Mondal & Tewari, 2007). Examples include the measurement of soil pH and humidity. One of the most significant soil properties to measure is its EC (Electrical Conductivity). According to Grisso et al. (2009), soil EC correlates with other properties such as soil texture, cation exchange capacity, drainage conditions, organic matter level, salinity and subsoil characteristics.

Variable rate technology is used to implement variable rate application. Variable rate application is one of the most fundamental concepts of precision farming. It is a technique where the application rate of fertilizers or seeds or other substances changes according to the properties of specific locations in a field. The implementation of variable rate application is dependent on the positioning system of the tractor and the electronic control capabilities of its implements.

Information transmission in this context refers to the transfer of data from the work machines. Most modern agricultural work machines use the ISO 11783 communication protocol (introduced in section 2.6), so the format of data transfer in work machines has become pretty well standardized. Of course there are still a lot of manufacturer dependent differences in different tractors and implements. A standardized and reliable technology to transfer the data from the ISOBUS network to some external system is yet to be decided. In addition to the ISOBUS data, data from the positioning system, which is usually an external device, must also be transferred.

The benefits of precision farming are increased yield and smaller negative environmental impact. Especially in Europe the need for precision farming has increased significantly in recent years. This is partly due to the fact that farms in most Western European countries are quite small compared to the sizes modern and expensive agricultural technology would be capable to handle (Bliss, 2015). Since the competition is quite tough, relying on old technology is often not feasible. Thus it is necessary for most farmers to maximize the yield from their small amounts of land. Environmental awareness has also increased in recent decades, resulting in stricter regulations for farmers to abide to.

2.4 Fleet management

Fleet management refers to a variety of considerations that are involved in the effective managing of a fleet consisting of any mobile vehicles, like cars, ships, aircraft, trains as well as agricultural work machines. Sørensen & Bochtis (2010) define agricultural fleet

management as the decision-making of a farmer or a machine contractor concerning, for example, resource allocation, scheduling, routing, real-time monitoring of vehicles and materials, supervision of the use and maintenance of the machines and the administrative functions associated with these. Various fleet management tools are used as support for these decisions.

In large farms, effective fleet management is a requisite for the effective execution of the various farming operations. The central task of fleet management is related to the logistics involved in these farming operations. A very basic example of the objectives of fleet management in terms of logistics is to minimize the operation time of each machine while completing the necessary tasks, thus reducing the fuel consumption and wearing of the machine as well as the working hours of the drivers. Agricultural work machines are large investments, making their optimal acquisition and utilization an important priority (Sørensen & Bochtis, 2010).

However, due to the biological nature of agriculture, there are a lot of uncertainties and dynamic considerations to account for. Large scale harvesting is a multi-stage operation requiring the sequenced usage of many different agricultural machines, like harvesters, transport trucks and unloaders. Planning the harvest operation can be a complex task which involves the scheduling and route planning of all these machines while considering factors like weather, varying yield size, customer needs, individual machine performance and everything else that may affect some part of it during any point of the harvest. Traditional fleet management tools are used to create a plan beforehand with off-line data. Similarly to precision farming, the actual execution of a previous cycle is compared to its plan, and this information is used to fine tune the next cycle. However, all these different variabilities related to the different actors involved in fleet management quite clearly indicate a need for dynamic plan revising based on on-line data. This is featured in the conceptual model for fleet management presented by Sørensen & Bochtis.

While the logistics point of view is the most dominant when discussing fleet management, machine maintenance can be counted as an equally important part since the proper operation of the machines is a requisite for the effective execution of the scheduling and routing plans. Work machines often have a fixed periodic schedule for inspection and maintenance, but this is not always enough to prevent malfunctions during operation. Depending on the data available from an individual machine, online data from the work machines could possibly be utilized to warn about and prevent these malfunctions, thus reducing larger and more expensive repair operations and giving the farmer more leverage in creating a replacement plan.

2.5 Work machines

Though modern farms may involve many different kinds of technology ranging from weather stations to building automation to IT-systems, agricultural work machines refer to the tractors and the various implements attached to them that are used to perform the

actual farming operations. There is little fundamental variability in tractors since their function has always been to mobilize the implements attached to them. There are many different implements each designed for a specific field operation, like tillage, seeding, fertilizing, harvesting, baling etc. A single implement may also be capable of handling more than one of these tasks.

The work machines a farmer uses rarely come from the same manufacturer. The tractor can be from a different manufacturer than the implements, and even the different implements used by the same tractor can all be from different manufacturers. This has long been the case, so the standardization of the connections between tractors and different implements has been necessary.

These work machines have experienced a rapid evolution from purely mechanical completely human controlled devices to electronically controlled almost fully or completely automated systems with sophisticated user interfaces. This raised interoperability issues between agricultural machinery from different manufacturers since the implements were now controlled remotely from the tractor and there was no standard method specified for this communication. Currently, the ISO 11783 communication protocol has become the accepted standard in most modern agricultural machinery.

2.6 ISO 11783

ISO 11783 (often referred to as ISOBUS) is a communication protocol designed specifically for agricultural work machines. Motivation for its creation originated partly from the fact tractors and their implements have developed rapidly in the past years, with increasing amounts of electronics and automation becoming commonplace. For feasible interoperability between various tractors and implements from different manufacturers, a common communication standard was necessary. One of the aims of the ISOBUS standard is to enable the connection of different implements in a plug-and-play manner (ISOBUS Specification Part 1, 2006).

ISOBUS is partly based on an older standard, SAE J1939, which is a similar standard originally intended as a communication standard for cars and trucks. They are both based on the CAN 2.0B protocol and similar message frames, allowing both messages to be passed on the same bus (ISOBUS Specification Part 1, 2006). This is important, since many tractor buses use SAE J1939 for communication (Piirainen, 2014), and it allows electronic control units designed to meet the requirements of SAE J1939 to also be used with agricultural equipment (ISOBUS Specification Part 1, 2006).

ISOBUS is a large standard, consisting of 14 different specifications. The main purpose of the standard is to standardize the method and format of data transfer between sensors, actuators, control elements, information storage and display units and whatever electronic systems an agricultural work machine might have (ISOBUS Specification Part 1, 2006). In this paper, we don't go into much detail of the ISOBUS standard. This section introduces some of the most essential terms and concepts of the standard to give the

reader a general picture of the aspects related to the OPC UA server for ISOBUS work machines.

2.6.1 Terms and concepts

An electronic control unit, or ECU, is an independent electronic item, the purpose of which is to implement a certain control function, like the nozzle control of a fertilizer applicator (ISOBUS Specification Part 1, 2006). ECUs are the main actors of an ISOBUS network, as they implement the actual functions to operate the agricultural equipment. A single control function can consist of multiple ECUs (ISOBUS Specification Part 1, 2006). There are some ECUs in the network with special roles, including the tractor ECU, task controller and virtual terminal.

The tractor ECU is a network interconnect unit designed to handle the messages between the tractor network and the implement network if they are included in the same system. Messages from the implement network are interpreted by the tractor ECU and communicated forward to the ECUs in the tractor network and vice versa (ISOBUS Specification Part 1, 2006).

The virtual terminal is an ECU consisting of a display and several buttons that give the operator an interface to a tractor or an implement. The actual interface for the virtual terminal is downloaded from the implement, and communication between the terminal and the implement is handled through standardized ISOBUS messages (ISOBUS Specification Part 1, 2006).

The purpose of the task controller is to provide scheduled control for the control functions of an implement, and to collect data from these completed control functions for evaluating how well they were executed according to the plan. Tasks are given to the task controller as a standardized XML-file, which is created with an FMIS. The task file contains complete information about what the task is, when and how it should be done, how much of each substance to use etc. When the task file is received, the task controller sends control messages to the different ECUs required to execute the defined task.

The most important concept from the point of view of this thesis is the DDOP (Device Description Object Pool). It is formed as an XML-file which can be either distributed separately by the device manufacturer or it can be exported directly from the device through the ISOBUS network. The purpose of the DDOP is to provide all information about the device necessary for task planning and execution for the device. This information includes the type and serial number of the device, the list of separate smaller entities forming the device such as tanks and controllers (called device elements), the geometry of important parts of the device and the supported process data variables of the device. This information is needed by the FMIS in planning the task file, and by the task controller in executing it (ISOBUS Specification Part 10, 2009).

2.6.2 Communication and overview

A typical ISOBUS network consists of a tractor connected to various implements. In addition to normal tractor functionality like brakes and transmission, the tractor bus also holds the important ECUs mentioned above; the tractor ECU, the task controller and the virtual terminal. To the tractor, various rear- or side-mounted implements can be attached. They contain network interconnect units which are used to connect multiple network segments with different architectures to each other, as well as various ECUs to execute the control functions of those implements.

Messages in the ISOBUS network consist of one or more CAN 2.0B data frames. In addition to the data it is carrying, each frame contains an identifier. The identifier contains information such as the frame priority, its source and destination addresses, and the PDU (Protocol Data Unit) format of the frame, which is used to determine whether the frame is sent to a specific destination address or using the group extension format, detailed in (ISOBUS Specification Part 3, 1998). From different components of the ID, which are also detailed in (ISOBUS Specification Part 3, 1998), the PGN (Parameter Group Number) of the frame is determined. The PGN is used to determine the actual type of the message, such as request, acknowledgment or data transfer. If a message is formed of more than one CAN frames (then called a multi-packet message), the PGN of each frame is the same.

3 OPC Unified Architecture

3.1 General

OPC is a communication standard designed for vendor-independent data exchange in an industrial environment, especially in SCADA (Supervisory Control And Data Acquisition) and HMI (Human Machine Interface) applications. A basic use case for OPC is, for example, to continuously collect sensor and other data from one or more devices such as a PLC and pass this data between the devices and an HMI. OPC Classic has become widely accepted and it has gained a firm foothold in many industries, despite some of its shortcomings, such as dependence on windows platforms and problems in remote communication due to COM and DCOM technologies (Mahnke, 2009). OPC UA is an improved implementation of the ideas behind OPC Classic, attempting to fix its shortcomings and to expand the usefulness of the architecture with improvements such as the object-oriented address space model, platform-independence, service-oriented architecture, secure and access controlled internet communication, extensible information modeling and meta information (Mahnke, 2009). OPC UA has been under active development in the last few years and while it is generally accepted that OPC UA has future potential and is capable of the tasks required of OPC Classic and more, its adaptation is still rather minimal compared to OPC Classic.

The complete OPC UA specification is quite large. This chapter introduces the basic principles of OPC UA and the specifications and features necessary from the perspective of this thesis.

3.2 Client-server model

Servers and clients are the most fundamental building blocks of OPC UA, since OPC UA is implemented as a system following the client-server model. OPC UA servers and clients are software applications usually developed with a UA SDK (Software Development Kit).

UA servers are applications that expose information from a device, such as the sensor data of an assembly line or a mobile work machine. A UA server supports one or more information models, which define how the information exposed by the server is typed and classified. The actual representation of this information is called the servers address space. Both of these concepts are opened in more detail in the following chapters.

UA clients are applications that connect to UA servers to operate on the information exposed by them. Clients are used, for example, to find information from the server's address space, read and write server data, subscribe to certain changes or events and to call server methods. This communication between clients and servers is handled by services, which are a part of the OPC UA specifications (OPC UA Specifcation Part 4, 2012). The relevant services for this thesis are introduced in section 3.4.

Though UA servers and clients are clearly functionally separate entities, they don't have to be separate from the application perspective. A single application can hold both a server and a client. The aggregating server, which is introduced in section 3.5, is an application consisting of one UA server and multiple UA clients connected to different servers.

3.3 Address space and information modeling

The Address Space, defined in OPC UA Specifcation Part 3 (2012), is the actual representation of the information that a UA server exposes. It is formed by nodes and references, which can be explored by a client. With a graphical client like UAExpert, the address space can be explored by its hierarchical references in the same manner as a normal folder structure. The default structure of the address space consists of a Root folder holding three main folders: Objects, Types and Views. An example of a simple address space is presented in Figure 1.

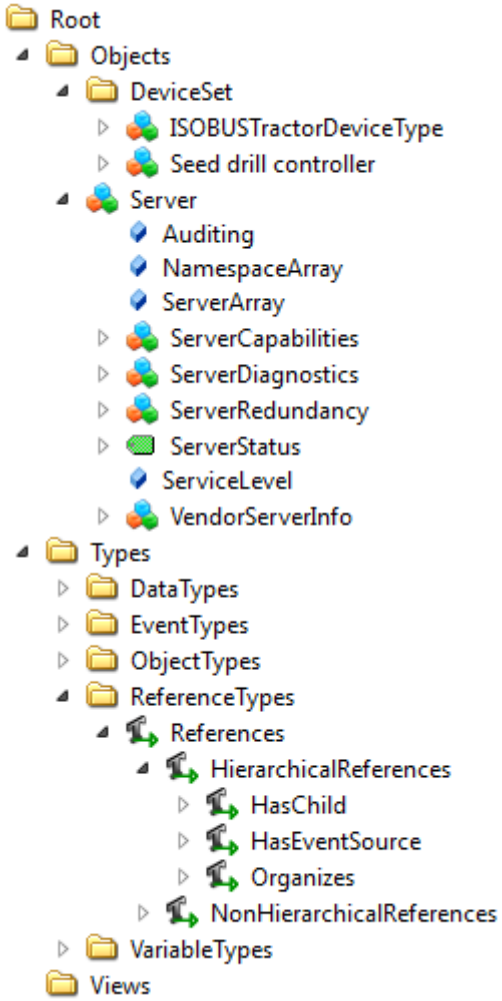


Figure 1. OPC UA address space. Everything except the DeviceSet folder and its contents belong to the default address space.

The Objects folder holds the instantiated node types of the server. That is, every device, sensor or whatever the server is exposing information from, is shown somewhere in the

Objects folder as a node or a group of nodes. This folder also holds a Server Object, which implements the Capabilities and Diagnostics extension to provide information about the server itself.

The Types folder holds the TypeDefinitions of the server as nodes. The purpose of this is to allow clients to access all type information of the server without beforehand knowledge. All nodes in the Objects folder must have a HasTypeDefinition reference pointing to a node in the Types folder.

The Views folder contains the entry points to the specified views of the server. From here, the desired excerpts of the address space can be browsed while the rest of the nodes remain invisible.

3.3.1 NodeClasses

To represent information more unambiguously, nodes are divided into NodeClasses, the most important of which are Objects, Variables and Methods (Mahnke, 2009).

Objects represent entities that can have variables and methods, both of which must always belong to an object (Mahnke, 2009). The purpose of objects is to group information, and they usually represent a complete entity like a work machine, or a certain component of a machine. Objects cannot by themselves represent a data value; values must always be represented as Variables.

Variables are used to represent values of different data types. Though the value of a Variable can originate also from some internal nodes of the server, Variables are used to connect the server to some real world data source, for example, the output of a temperature sensor. Variables are also used to express additional metadata for a node.

Methods relate to Objects in a similar way as they do in object-oriented programming. They are called by a client with the specified input arguments, and after the method is run, the specified output arguments are returned to the client. Methods can be used for many kinds of purposes, including calculation operations based on some current or historical Variable values and passing commands to the underlying machine, for example to start a motor.

One less used NodeClass is the View. On many servers they are not implemented at all, but when looking at aggregating servers especially from the point of an FMIS, they appear to hold a lot of potential. Views are used to restrict the visibility of the server's Address Space. This allows the user to pick a specific view for a specific task. For example, a "Process Data View" could be implemented, which would show from every object only certain variables related to exposing process data, while hiding everything else. The address space size of some UA servers can possibly reach hundreds of thousands of nodes, at which point browsing for information by hand becomes nearly impossible.

3.3.2 References

References are used to relate nodes to each other. Even though technically a reference can exist by itself, it cannot be accessed on its own and it has no meaning without a source and a target node, since their only purpose is to define the semantics of how different nodes are connected (Mahnke, 2009). For example, the sensor data nodes (Variables) of a machine (Object) could be connected to the machine with references of type HasComponent originating from the machine Object to each of the Variable nodes (more information on types and some examples of them are introduced in section 3.3.4).

While there is no point in having references without nodes, there is likewise little sense in having nodes without references leading to them. This is because in addition to defining the semantics of the structure between the nodes, another important function of references is to allow browsing the address space of the server. If the NodeId of a desired node is not known beforehand, it cannot be accessed without a reference leading to it.

3.3.3 Attributes

Attributes are used to describe a node. They do not represent metadata related to whatever the node itself is representing, but rather information about the node relevant to the server and clients. All nodes have a set of attributes, some of which are common for all nodes regardless of its NodeClass. The common attributes for all nodes are listed in Table 2.

Table 2. Common attributes for all nodes (Mahnke, 2009)

Attribute
NodeId
NodeClass
BrowseName
DisplayName
Description
WriteMask
UserWriteMask

The most important attribute is NodeId, which identifies the node. All NodeIds in a server must be unique. NodeIds are used to refer to nodes in service calls made by clients. NodeIds consist of a NamespaceIndex and an identifier. NamespaceIndex represents a naming authority of the current type of node. Each server holds a list of the NamespaceUris on the server, each representing a part of the server's information models. For example, the NamespaceUri of the OPC UA base information model is "http://opcfoundation.org/UA/", and it always has the NamespaceIndex 0, regardless of the server. The order of other NamespaceIndices is not fixed (Mahnke, 2009). The identifier part of the NodeId can be either a numeric value, String, GUID (Globally Unique Identifier) or a byte string (Mahnke, 2009).

BrowseName is the name of the node for browsing purposes. BrowseNames consist of the NamespaceIndex of the node and the name string. DisplayName is the name of the node as shown to clients. It consists only of a localized string. Description is also a localized string, and can contain additional information about the node. WriteMasks and UserWriteMasks define which attributes of the node are writable, respectively for any client and only the current user.

Depending on the NodeClass, there are a lot of attributes that are not common between all classes. An important one of these is the Value attribute of Variables. This attribute exposes the actual value of whatever the variable is representing, for example the output of a sensor. Values can be of many data types, and they can be either scalars or arrays of one or more dimension. These are defined by additional attributes of the NodeClass Variable.

3.3.4 Types

The ability of OPC UA to expose detailed information about different systems can be largely attributed to Types. All nodes (except methods) and references must have a type definition (Mahnke, 2009). Type definitions are used to describe the semantic as well as the underlying structure of a node, to which all instantiated nodes must adhere to.

The OPC UA base information model provides basic type definitions, which can be used to describe almost any system, but are rather abstract in nature. However, the power of OPC UA information modeling comes from the ability to create custom information models for a specific purpose. The types of the base information model can be subtyped to represent more detailed and specific entities. For example, a temperature sensor could have its own “TemperatureSensorType” type definition, or it could be an instance of a more general “SensorType” type.

Type definitions can be either simple or complex. A simple ObjectType only defines the semantic of the object, whereas a complex ObjectType defines an underlying structure of nodes for the object (Mahnke, 2009).

Object types

On the top of the object hierarchy of the OPC UA base information model is the BaseObjectType. All other object types are inherited from this base type.

Variable types

Variables are divided in two distinct types which should be used in different contexts: Data Variables and Properties. Data Variables are used to expose actual changing data from the underlying system, like sensor readings, states of a device etc. Data Variable types can be complex, though their children are limited to other Data Variables and Properties.

Properties are always simple, which means that they are always the leaf nodes of a hierarchy. They are used to expose metadata and to describe nodes where attributes are not

enough. Common uses for properties are, for example, to define the unit of a value exposed by a DataVariable, or to tell how many times a node in the address space has changed.

One relevant standard property from the point of view of this thesis is the ServerArray, located under the Server object. The ServerArray contains a list of the ServerUris of the server. A ServerUri is a unique string identifier of an OPC UA server. The first ServerUri in the ServerArray always belong to the local server, while the rest of the ServerUris point to other OPC UA servers to which the server may be connected to. The indices of the ServerArray can be used to refer to nodes which are located on another server.

Reference types

Even though references are not nodes and they cannot be accessed directly, ReferenceTypes are exposed as nodes in the address space the same way as all other type definitions, allowing clients to access information about the references. ReferenceTypes also have all the common attributes listed in Table 2, as well as some additional attributes.

ReferenceTypes are divided into hierarchical and nonhierarchical references. Hierarchical references contribute to the hierarchical structure of the server address space (though they can still lead to loops), while nonhierarchical references do not. An example of a nonhierarchical ReferenceType is the HasTypeDefinition reference, which points from a node instance to the TypeDefinition node of that node. Some examples of hierarchical ReferenceTypes are HasChild, HasComponent and Organizes.

3.3.5 Standard extension information models

In addition to the base information model defined by OPC UA Specification parts 3 and 5, the OPC UA specifications define some standard extension models, as well as some domain specific companion models, created by other authors (Mahnke, 2009). The standard extension information models and the specification part where they are defined are listed in Table 3.

Table 3. Standard information model extensions (Mahnke, 2009)

Information model	Specification
Capabilities and Diagnostics	(OPC UA Specifcation Part 5, 2012)
Data Access	(OPC UA Specifcation Part 8, 2012)
Historical Access and Aggregates	(OPC UA Specifcation Parts 11 and 13, 2012)
State Machine	(OPC UA Specifcation Part 5, 2012)
Programs	(OPC UA Specifcation Part 10, 2012)
Alarms and Conditions	(OPC UA Specifcation Part 9, 2012)

All of the listed extensions include type definitions for nodes that ease the implementation of the functionality that information model is designed for. The Capabilities and

Diagnostics model contains information about the server, its clients and the services used in it, from the perspective of either the whole server, a single session or a single subscription. Data Access defines variable types and properties, which assist in representing data in a clearly defined way. The Historical Access and Aggregates model extends on the Capabilities and Diagnostics model. It defines how historical time series and event data are represented and accessed, and provides ways to expose aggregated data. As the name suggests, State Machine provides and information model for state machines. This model is used by both Programs and Alarms and Conditions. Programs are used for more complex implementations than what could be achieved with methods only. Programs are long running and stateful functions that can be used, for example, to run control operations. Alarms and Conditions uses state information to define event-based alarms and conditions.

The domain specific models are not part of the standard OPC UA base information model specification, although some of them are certified by the OPC foundation and added to a companion specification (Mahnke, 2009). Some examples of these include information models for MTConnect (OPC UA Companion Specification, 2013a), Analyser Devices (OPC UA Companion Specification, 2013b) and ISA-95 (OPC UA Common Object Model, 2013).

The most important companion information model for this thesis is OPC UA for Devices (OPC UA Companion Specification, 2013c), which provides a unified way to model devices as Objects and to expose all its parameters. UA Devices is a very general model, which is designed to present most devices regardless of their underlying protocols. For more domain-specific implementations, the types provided by UA Devices can be subtyped to represent a more specific device type accurately. In the previous thesis, an information model intended specifically for ISOBUS work machines was designed based on the Devices information model. This OPC UA for ISOBUS model is used as a basis for the aggregation software of this thesis, and it is explained in more detail in section “3.6 Previous Work”.

3.4 Services

Services, defined in OPC UA Specifcation Part 4 (2012), are calls directed at UA servers by clients to handle the communications between them. Services are meant for a specific task, though a server may define custom implementations for some services (for example, this is useful when implementing an aggregating server). OPC UA specifies a lot of services, of which only a few are immediately relevant for the development of an aggregating server. These services are introduced briefly below.

Probably the most fundamental service is Browse, which is used to navigate the address space of a server. The Browse service is called for a starting node with some specified filters, and it returns a list of nodes connected to the starting node, according to the specified filters (for example, only nodes connected to the starting node with a specified ReferenceType could be returned).

While Browse is used to find nodes, it only returns the attributes from each node that are necessary to fill up the browse tree. To actually access and manipulate the data or metadata of those nodes, two new services are required: Read and Write. Read is used to read the rest of the attributes of a node. Though it can be specified which attributes are returned, the most usual use case for read is to always return all attributes of a node. The working principles of Write are the same as those of Read, but instead of reading, the specified attributes are overwritten.

When it is necessary to continuously monitor a node instead of a single Read operation, the Subscribe service can be used. There are a lot of other services that are needed by Subscribe, but in a nutshell, it is used to monitor either a change in the Value attribute of a Variable, an Event, or an aggregated Value. When a client subscribes to any of these, a Monitored Item for the current type (Value, Event or aggregated Value) is created. A single subscription made by a client can hold multiple instances of all of these Monitored Item types (Mahnke, 2009). Each monitored item has a sampling rate, which defines how often the server checks for value changes in the items. When a value change or an Event occurs, the server sends notifications to the client, and the value is updated.

Many OPC UA servers collect historical data from the values of certain nodes into a database. This database can be implemented in whatever way is found feasible, as long as the server supports the necessary services to allow clients to read data from the database regardless of the implementation. An example of collecting history data from UA nodes to an SQL-database is presented by Asikainen (2013) in his master's thesis. With the HistoryRead service, a client can retrieve the values for one or more nodes by supplying the NodeIds of the required nodes and the desired time interval from which the values for the nodes are then returned (OPC UA Specifcation Part 11, 2012).

3.5 Aggregating servers

An aggregating server is a server architecture concept in OPC UA. The concept is briefly introduced by Mahnke (2009) and the basis for it is built into the OPC UA specification, but as of yet, to the best knowledge of the writer, there are no generally accepted specific solutions for them. Essentially it is a system of OPC UA servers and clients that allows concentrating some or all information from multiple servers to a single aggregating server. The structure of an aggregating server is exemplified in Figure 2.

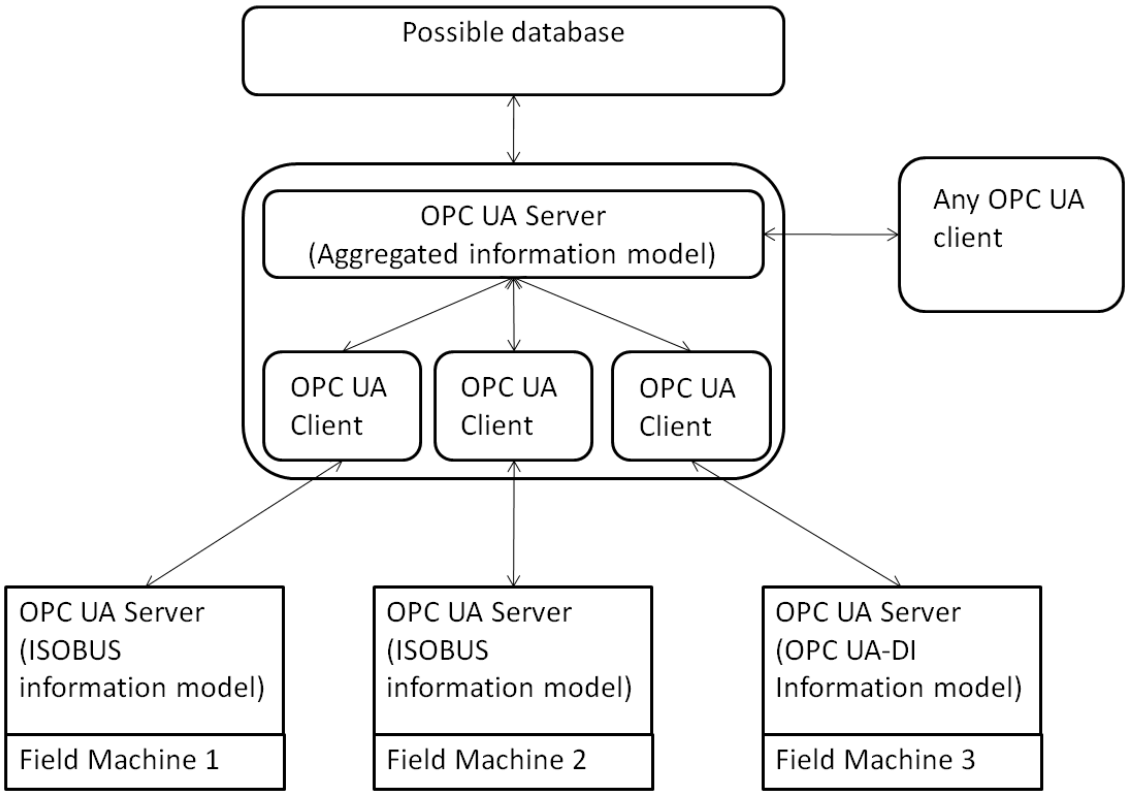


Figure 2. Example of an aggregating server architecture

The ability of an aggregating server to concentrate data from other UA servers is achieved via internal clients embedded to the aggregating server. For each underlying server to be aggregated there exists an internal UA client connecting to it. The client is used to access the address space of the underlying server via the standard UA services used in client-server communication. (Mahnke, 2009). The desired parts of the underlying address space are mapped to the address space of the aggregating server. These address space chunks may be instantiated to the aggregating servers address space exactly as they appear on the underlying server or they may be transformed to a different configuration during mapping. The address spaces of the underlying servers can vary greatly, so for the aggregating server to be able to offer a unified and concentrated view of the servers it is aggregating, the relevant information from these underlying address spaces must be made to conform to the structure of the aggregating address space. The aggregating server may also condense the data from multiple underlying nodes to a single aggregated node, displaying, for example, the average or some other calculated value from these nodes.

From a client's perspective, an aggregating server is like any other UA server. It responds to the same service calls and the information from the underlying servers is directly obtainable from the aggregating server's address space without prior knowledge of the underlying servers. An aggregating server may also itself be aggregated. This offers a lot of flexibility enabling OPC UA to be used in multi-level system integration where each level can aggregate previous levels. These aggregating servers can then be

used as interfaces to both monitor and control lower level functionality. Even though clients may not be able to tell the difference between an aggregating server and a regular UA server, the functionality of certain services between the service call and returning the results have to be modified for the aggregating server to relay the content of these calls between the client and the underlying servers.

The most basic service implementations an aggregating server needs to modify are read, subscribe, and, depending on the use case of the server, write. For example, when a client wants to read the value attribute of a specified variable node in the address space of the aggregating server, this service call must be relayed to the corresponding node in the underlying server to fetch the actual value, which is then used to overwrite the old value of the aggregating node and returned to the client. Depending on how and for what purpose the aggregating server is used, the requirements for modifications and how they can be implemented may vary greatly. In any case, there must be a way to connect the nodes from the aggregating address space to the corresponding nodes in the aggregated address spaces.

3.5.1 Address space mapping

A typical use case of an aggregating server is such that it connects to multiple servers with varying address spaces and different information models, extracts the relevant nodes from these address spaces and exposes the extracted content from all underlying servers in a unified aggregating address space. This may be done manually, but most likely some automatic mapping mechanism is required.

A mapping mechanism must be able to browse through the remote address space, identify the correct nodes to be aggregated and to create the corresponding nodes to the aggregating address space according to some mapping rule set while maintaining a connection between these nodes. If the information models and the structure of the address space of both the underlying and the aggregating servers are known beforehand, creating fixed mapping rules for them is fairly simple. However this is an unlikely case, since the underlying servers may have different information models and the address space structures of similar devices even in the same server may have large differences.

The OPC UA specification defines no standard method for this address space mapping problem, nor has there been a lot of research on the matter. In one of the few studies made, Großman et al. (2014) proposed a model where the mapping rules are included in UA information model extensions. In his model, both the aggregating server and each underlying server contain an information model extension defining the rules required for address space mapping.

Großman's information model for the aggregating server defines types that are mainly used to keep a list of all aggregated servers and available servers yet to be aggregated. The information model for the underlying servers contain the actual mapping rules. There are two separate types for this, one specifying the rules for mapping types, and

another one for instances. Type mapping rules are used to identify semantically identical types with different nodeIds or BrowseNames from the underlying servers. The information model also contains two methods, ExportTypeDefiniton and ImportTypeDefinition, which are used to serialize the necessary information of a type in an underlying server to establish the same type in another server.

One of the strengths of Großman's model is that the mapping rules are included in the UA information models, thus requiring no external mapping rules. However it may not always be feasible to implement the extension model to all underlying servers. In these cases, the mapping rules must be defined via a separate aggregation configurator. Großman's model seems very useful in identifying types and mapping them against each other. However, he states in his paper that "the corresponding references between the proxy nodes are also maintained just as they are maintained on the underlying servers, so that the structure information of the underlying address space is preserved". This makes his solution infeasible if the aggregation procedure involves the structural transformation of the underlying address space.

3.5.2 Commercial implementations

Aggregating servers are often very use case specific, so a general implementation that's easily usable in most circumstances is a difficult design task. However, there are some commercially available implementations designed for a specific purpose but usable in most environments.

One example of these is OPC UA Historian by Prosys (https://www.prosysopc.com/products/opc-ua-historian/). It is an aggregating server which is designed to store historical data from the underlying servers to an SQL database. The stored history data can be accessed via an OPC UA client or directly from the database with an SQL client. Prosys's Historian is heavily based on the master's thesis by Asikainen (2013).

Another example is the UaGateway by Unified Automation (https://www.unified-automation.com/products/wrapper-and-proxy/uagateway.html). It is an aggregating server that acts as a wrapper for classic COM/DCOM based OPC servers and OPC UA servers. It can connect to multiple UA and OPC Classic servers at the same time and allow access to these with either a UA or a OPC Classic client. It can also tunnel COM/DCOM connections through a UA connection, offering more secure access to OPC Classic servers.

3.6 Previous work

This thesis is based on previous work done by Piirainen (2014). In his thesis, he designed an OPC UA information model (OPC UA for ISOBUS) for a tractor/implement ISOBUS network, and implemented a working prototype UA server which was tested both in laboratory conditions as well as with a real machine. The model exposes the

internal states of the tractors subsystems, as well as the process data from the tractor and its implements, the latter of which is defined by the implements DDOP.

OPC UA for ISOBUS defines object and variable types for the implements DDOP, the tractor and for the GNSS (Global Navigation Satellite System) of the tractor. The ISOBUS information model is an example of the versatility of OPC UA information modeling due to sub-typing. The object types for both the devices and device elements from the DDOP, the tractor and the GNSS are all inherited from object types of the OPC UA Devices model. OPC UA for ISOBUS uses the full extent of OPC UA Devices while defining its own more specific types for different devices.

For this thesis, the most relevant part of the ISOBUS information model and UA server is the DDOP. The object types for the DDOP are shown in Figure 3.

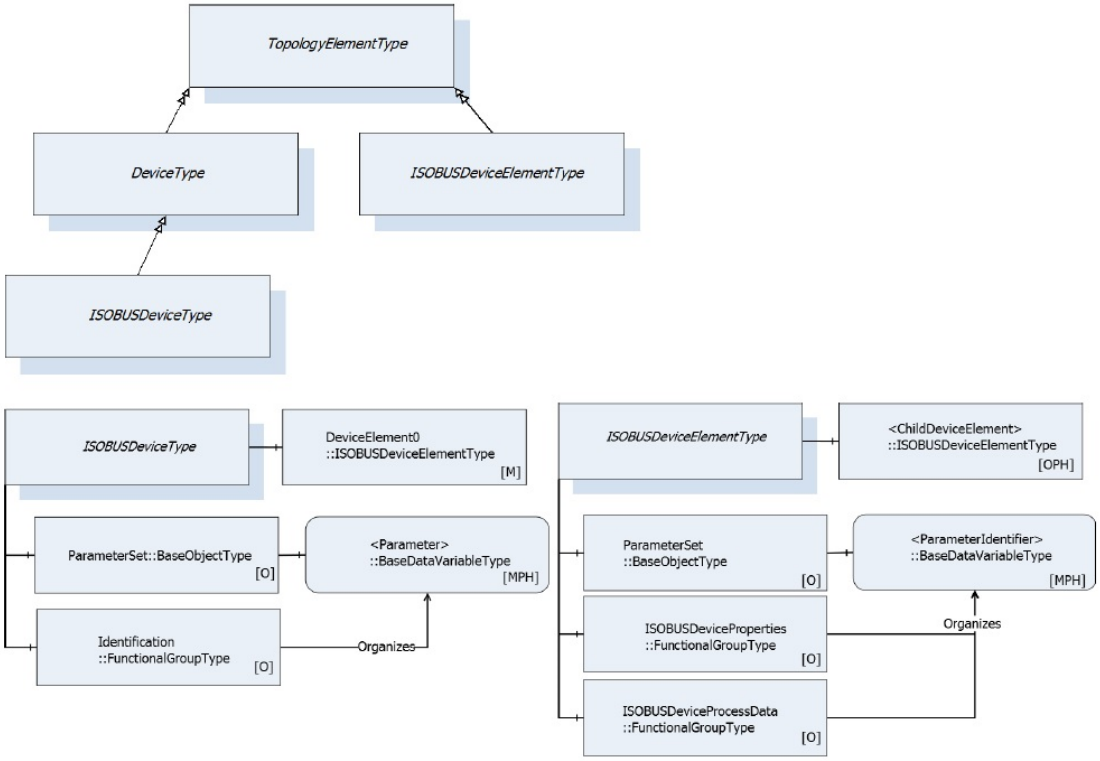


Figure 3. UA object types for the implements DDOP, adapted from Piirainen (2014). The arrow with two triangular heads represents a HasSubtype reference, and the arrow with the short perpendicular line as its head represents a HasComponent reference

The ISOBUSDeviceType represents the implement from which the DDOP originates. There is only one of these per DDOP, since each implement has its own DDOP. Each ISOBUSDevice object has at least one child ISOBUSDeviceElement object, but usually there are more of these. ISOBUSDeviceElementType represents the smaller items of the device to which its functionality can be divided to. Both of these types have a ParameterSet object as a child, which contains all the DataVariables of each device or device element. These DataVariables represent different characteristics of the device or device element, so for clarity, they are divided further into functional groups (with UA

objects of the type FunctionalGroupType) according to what kind of data they represent. ISOBUSDeviceType only has one functional group, but the DataVariables from device elements can be divided into multiple groups, such as the process data of the device element, its properties or its physical dimensions (Piirainen, 2014).

ISOBUSDeviceType has only one parameter, which is its NAME. NAME is a unique eight byte long identifier of an ECU, which contains information about the ECU, such as its control functions, manufacturer and serial number. The information model defines a NAMEType, which is used only for the variables exposing the NAME of an ECU (Piirainen, 2014, ISO, 2006).

The parameters for ISOBUSDeviceElementType have their own variable types defined in the information model. These are shown in Figure 4.

Figure 4. Variable types for ISOBUSDeviceElementType, adapted from Piirainen (2014)

The DDOP file specifies a number of process data items and properties for the device, which are represented by the types inherited from ISOBUSAnalogItemType and ISOBUSDiscreteItemType. All of these types have as their properties a designator, which is just a string describing the process data item or property, a data dictionary identifier (DDI) and a data dictionary entity (DDE) name. A data dictionary is a file which lists all process data items and properties that ISOBUS devices use, and the necessary information about them, like their units, minimum and maximum values etc. The

DDI is the index number of the current process data item or property in the data dictionary file, and DDEName is the name corresponding to the index number specified in the data dictionary file. ISOBUSAnalogItemType is defined for DDEs that have a continuous data type (double), and ISOBUSDiscreteItemType for DDEs that have a discrete data type (enumeration).

For the data from the sub-systems of the tractor or from the GNSS, the following object types were defined: ISOBUSTractorDeviceType and ISOBUSGNSSDeviceType. Both of these types expose their data categorized in functional groups in the same way as ISOBUSDeviceType and ISOBUSDeviceElementType. ISOBUSTractorDeviceType exposes the data from the subsystems of the tractor, and ISOBUSGNSSDeviceType exposes the positional data from a GNSS receiver, which is usually an external or separate device from the tractor. The parameters of these object types are exposed as BaseDataVariableTypes. Both of these object types are inherited from DeviceType.

The information model also defines types for to the SAE J1939 communication network. It defines the J1939DeviceType, which is also a subtype of DeviceType. Like the other types inherited from DeviceType, J1939DeviceType also has a parameter set which is grouped into functional groups. However, the functional groups for J1939DeviceType use their own J1939FunctionalGroupType. It differs from the basic FunctionalGroupType in that it has a PGN-property. J1939FunctionalGroupTypes correspond to a J1939 message, which is identified by the PGN. The parameters for J1939DeviceType are exposed as BaseDataVariableTypes.

The prototype UA server is designed to work in a plug-and-play manner for implements. When the server is running and an implement is connected, a series of messages are sent back and forth between the server and the bus, which ultimately results in the transfer of the DDOP from the implement to the server. The DDOP is parsed by the server, and the according objects and variables representing the implement are instantiated to the server address space. After this the bus keeps continuously sending process data values from the implement, which are used to update the values of the corresponding nodes in the server.

4 Requirements

This chapter defines the requirements for the aggregating server as well as specifies the complete system it needs to interact with. The requirements are derived from the point of view of data acquisition in farm management, especially with regards to the needs of precision farming and fleet management. Section 4.1 defines the actors and components of the system of which the aggregating server is a part of. Section 4.2 defines the use cases concerning the system and the functional requirements for them. The last two sections specify additional requirements for the system.

4.1 System definition

The system that is the focus of this thesis is an aggregating OPC UA server that is connected to multiple agricultural work machines. Each of these work machines are running their own OPC UA server which is continuously exposing the available data from them, like process data about the execution of field operations or diagnostics data from the internal subsystems of the machines. The aggregating server communicates with these work machine servers, monitoring and storing the data available from them.

The aggregating server may be a part of a larger FMIS or it may be an independent system to which one or more FMISs are connected to. Even though FMISs can be sophisticated and comprehensive software systems that are used in helping the farmer plan the different farm managing activities, the help usually comes in the form of off-line data about farm parameters and previous operations used in conjunction with a relational database for creating an efficient plan while complying with recommendations and regulations. The addition of an aggregating server is intended to provide on-line data and to provide a centralized means for the collection and storing of selected measurement data. This data can then be utilized, for example, to boost the performance of the FMIS, or it could be accessed by the farmer or any other stakeholder directly by a UA client. This thesis focuses only on the data acquirable from the work machines, but the aggregating server could be used to gather the data from most other data sources of a farm as long as they can be exposed by OPC UA servers. An example of a farm with an aggregating server and an FMIS is illustrated in Figure 5.

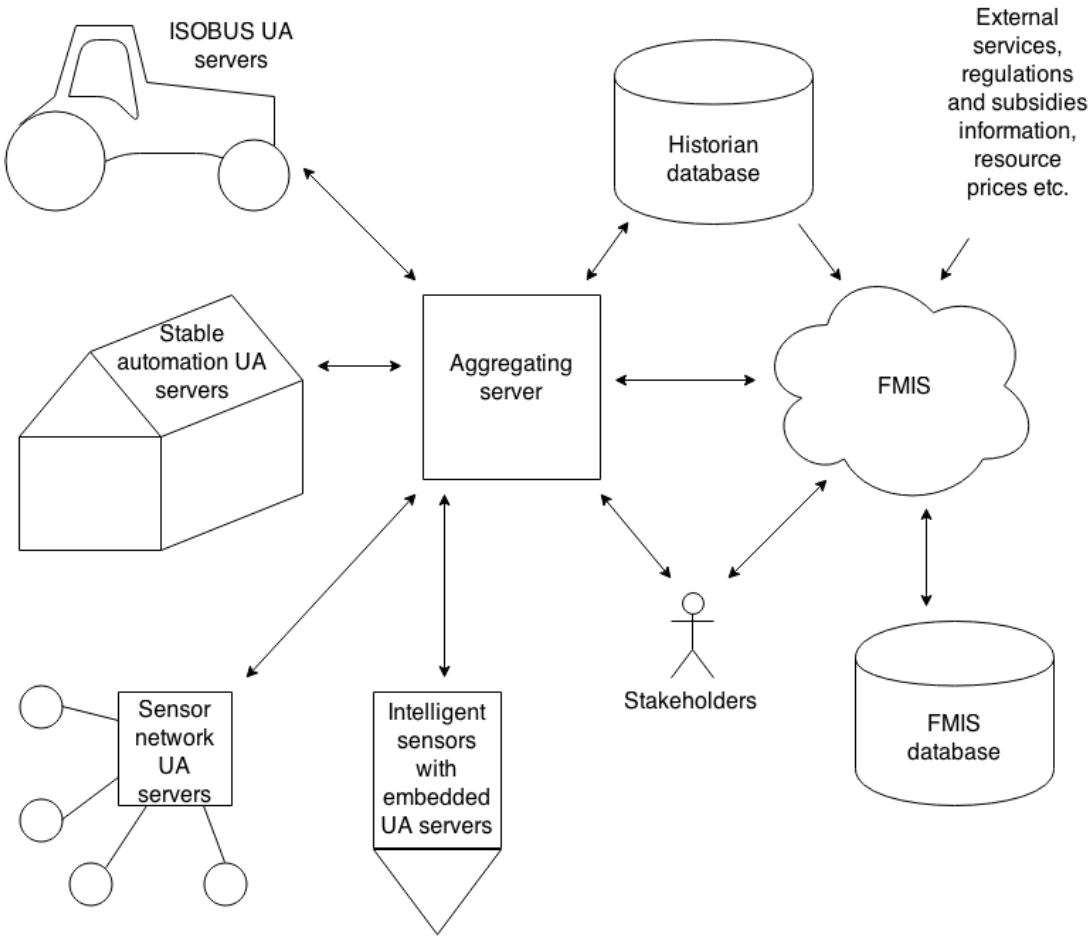


Figure 5. The aggregating server as a part of a farm.

There are many different stakeholders regarding the work machines and farm management, all of which may potentially be interested in the data gathered from the machines. They may also each have different needs and wishes with regards to the functionality of the aggregating server, and thus their views should be taken into account when designing a comprehensive system. In his thesis, Piirainen (2014) studied data access to agricultural work machines using OPC UA and defined the main stakeholders for such a case. Since this thesis focuses on the same application domain and methodology but just on a larger scale, the same stakeholders are applicable in both cases.

The primary stakeholder is the farmer. They are the main users of an FMIS since the decisions concerning farm management are ultimately made by them. For this reason, they require access to all relevant information related to the farm, including the data collected from the work machines as well as information related to the less technical and more managerial aspects of farm management. He is expected to be the most active user of the aggregating server. The farmer's potential lack of technical aptitude should be taken into account when designing how the data from the work machines is acquired and represented.

On a larger farm, there may be third party drivers hired by a contractor, both of which are considered stakeholders, though their needs in terms of data acquisition are signifi-

cantly smaller. The needs of the drivers are centered on what they need to successfully operate a single work machine and execute the planned field operations with it, but they do not require access to the data gathered by the aggregating server. The contractors may be interested in information that could help them improve their fleet management efficiency and employee performance.

From the work machine point of view there are three relevant stakeholders: the manufacturer, the maintenance provider and the dealer. A manufacturer is one of the many different enterprises that manufacture the work machines. He may be interested in data that could possibly be utilized in future machine development, though the information regarded as relevant may differ from manufacturer to manufacturer, and it rarely needs to be on-line data. Maintenance service providers may be independent actors, or the manufacturer or the dealer of work machines may also provide maintenance services. In any case, they may be interested in information that can help them in doing their job more efficiently. This information can include the internal and process data from the work machines, both off-line and on-line. Maintenance service providers could utilize on-line access to the work machines in remote diagnostics, which may make it faster to localize and fix problems on the site or perhaps to avoid a site visit completely.

Another group of stakeholders that should be considered are the IT-system developers whose responsibility is to develop the FMIS functionality not related to data acquisition. Since most current FMISs do not implement or utilize data acquisition from the work machines, the method of data acquisition should be designed to be general and expandable to ease integration with existing and future farm management software.

One final stakeholder is the administrator of the aggregating server. His responsibilities include the configuration of the aggregating server, managing the connections to the underlying servers, updating mapping rules if requirements for a certain information model change or if a UA server with a new information model is to be aggregated, and managing the server certificates and user access rights.

4.2 Use cases and functional requirements

In this thesis the main use case for the aggregating server is the automatic addition and address space configuration of a new device to the address space of the aggregating server. The user must also be able to read and write the data exposed by the underlying servers through the aggregating server. Proposed use cases for the aggregating server also include the storing and reading of historical data for selected data points, and uploading files from the aggregating server to the underlying servers. These use cases and their requirements are examined in sections 4.2.1 through 4.2.4.

Since this thesis is a direct continuation of the previous thesis and a part of the same overall project, the aggregating server must maintain the functionality defined for the individual OPC UA servers on the work machines. Use cases for the individual work

machines taken from the works by Piirainen (2014) and Ronkainen (2014) are briefly examined in section 4.2.5.

4.2.1 Adding a new work machine to the aggregating server

This is the most relevant use case for the aggregating server and for the focus of this thesis. This use case involves the detection of a new OPC UA server running on a device, reading its information models, and configuring its address space to the address space of the aggregating server according to the found information models. In addition to a work machine, the device may be a piece of some other farm related equipment, like an intelligent soil sensor or an automation system related to livestock farming. However, the scope of this thesis focuses only on the addition of ISOBUS tractors and implements running the UA for ISOBUS information model.

There are two ways to add a new work machine server to the aggregating server. The choice between these two depends somewhat on the scale of the farm or the amount of farms and the amount of work machines under the aggregating server. The aggregating server could be connected manually to each work machine server, such that when a new work machine UA server is running on the farm network, its address is entered to the aggregating server, which then connects to the underlying server and maps its address space. For small farms and irregular device additions this option may be feasible. However, on a larger farm where work machine additions are more frequent, the machines keep changing, or the state of automation is generally taken further, and also if the aggregating server is intended to be connected to devices from multiple farms, manually connecting to each new device is not desirable. The second choice is to have a local discovery server on the farm, to which new device servers are connected to during initialization. The aggregating server would then be connected to the discovery server, and whenever a new server is found, the address space mapping procedure is initialized. This configuration is illustrated in Figure 6.

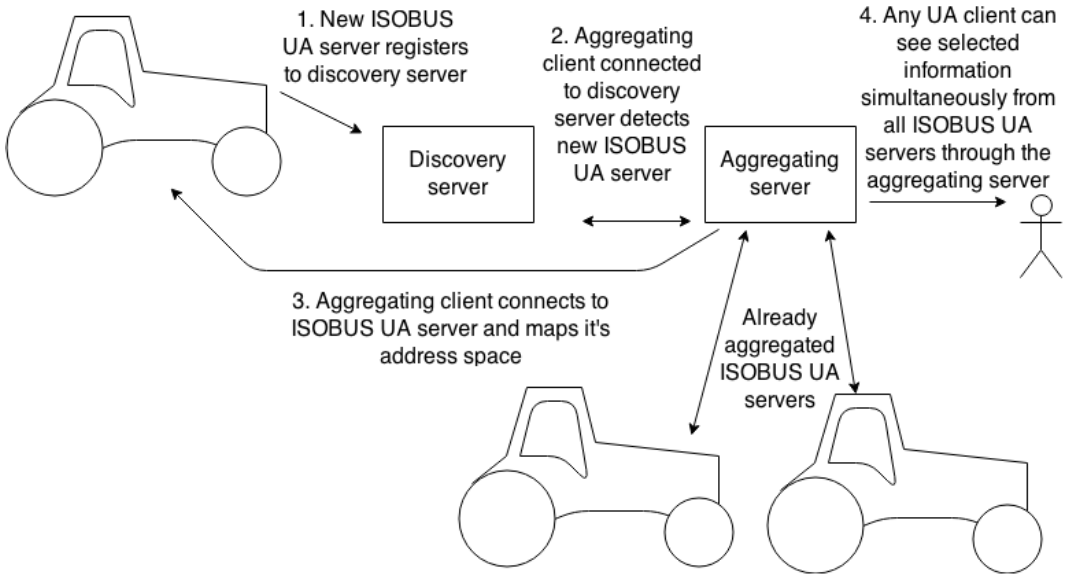


Figure 6. A new ISOBUS UA server is discovered and connected to an aggregating server already aggregating two other ISOBUS UA servers.

The OPC UA standard address space specifies a server object, which holds the namespaces of the information models supported by the server. When the aggregating server connects to an underlying server, it reads the information models and checks whether it has mapping rules for any of the information models found. If it does, the address space of the underlying server is mapped to the aggregating address space according to the rules defined for a specific information model. The aggregating server can have one pre-defined address space structure to which the underlying address spaces are mapped to, or multiple possible structures which can be chosen based on the information models of the underlying servers. It is also an option to bring a part of or the complete underlying address space to the aggregating address space as is, with no change in structure. The underlying address spaces can differ greatly from each other even if they are running the same information model (OPC UA for ISOBUS in this case). In addition, it must be able to map the address spaces from other OPC UA capable devices on the farm, many of which can have completely different information models.

4.2.2 Reading and writing data through the aggregating server

This use case is involved in maintaining the connection between the original nodes in each underlying server and their corresponding proxy nodes in the aggregating server. This is one of the most important functions of an aggregating server. When the farmer or any other stakeholder wants to read a certain data node through the aggregating server or subscribe to it, the aggregating server must relay this service call to the correct node in the correct underlying server, and then relay the service result to the client reading the address space, as illustrated in Figure 7. This same procedure must also be implemented for writing data to an underlying node through an aggregating proxy node.

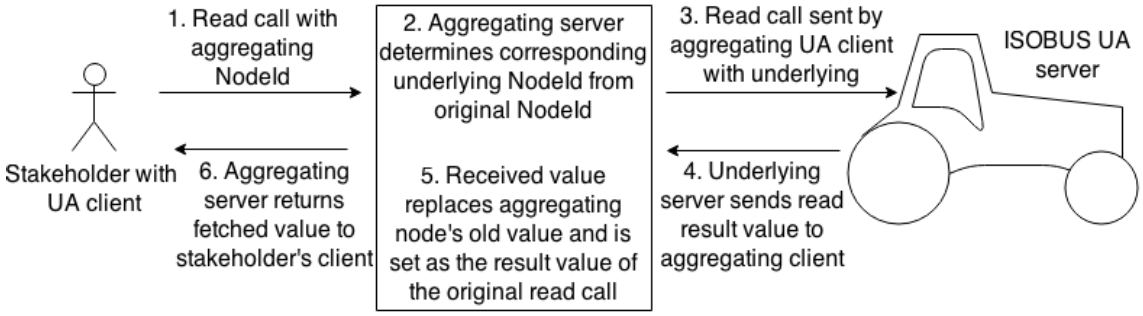


Figure 7. Service relaying procedure for a single read call.

4.2.3 Reading historical data from the aggregating server

When the aggregating server is running, a UA client can be used to read the on-line data from the currently connected and running underlying servers. However, in many cases it would be useful to be able to read historical data from some of the underlying servers through the aggregating server. For this, the aggregating server needs to store the values of selected nodes with a timestamp for each value. This historical data can be stored either to a proprietary data collection, a database or a short term memory buffer (OPC UA Specification part 11, 2012). For long term data storage, the last option is clearly

not viable. The storing of OPC UA historical data to an SQL database was demonstrated by Asikainen (2013) in his thesis.

Depending on the database, it may be easy to read the stored historical data directly from the database, and for some purposes this might even be reasonable. For example, the same database could be accessed directly by different software components for their own purposes. However, the data should also be accessible directly from the aggregating server with a UA client. For this, the aggregating server must implement the services defined by the OPC UA Historical Access information model and integrate them with the database. Thus, a client can use the HistoryRead service to get the values from the aggregating server without knowledge of the actual database where the data is stored.

4.2.4 Uploading files from the aggregating server to the underlying servers

This is a potential use case that would be mostly involved in using the aggregating server to host updates that would be downloadable to the underlying servers. These could be, for example, software updates to the underlying servers, or other software components connected to them. The OPC UA specification part 5 (2012) specifies a FileType node which could be used for file transfer between OPC UA servers. For the scope of this thesis, the main point of interest in this use case is the remote firmware update for ISOBUS machines. While this would in principle be possible from the UA perspective, the technical limitations of ISOBUS make this use case unfeasible.

4.2.5 Run time use cases for individual work machines

The following use cases are not for the aggregating server per se. These use cases are those chosen by Piirainen for the OPC UA for ISOBUS information model and server, and the ones involving ISOBUS from the CLAFIS project defined by Ronkainen (2014). Since the aggregating server is a continuation of the individual ISOBUS UA server project with a broader scope, the functional requirements for some of these cases concern the aggregating server as well.

Detect when an implement is started

The usage cycle for most implements is irregular in the sense that when they are used they are used intensely for a couple of weeks, after which they can be on a long hiatus. Naturally most malfunction situations and service requests occur close to the period of heavy usage. Information about the startup of an implement in the beginning of a season could be utilized by the machine manufacturer and maintenance service providers to time their resource allocations (Ronkainen, 2014).

The requirement for this use case is that the implements NAME is exposed by the UA server. This is done by the UA for ISOBUS server, so from the aggregating server's point of view the requirement is that the NAME exposed by the underlying server is also exposed in the aggregating address space.

Remote implement firmware updating and sensor malfunction situation
This is one of the use cases from the CLAFIS project. The case of remote implement firemware updating is already addressed in section 4.2.4.

The sensor malfunction situation refers to a case where one or more of the implements sensors malfunction, and the information about the system and the affected sensors should be accessed remotely by the service organization. The feasibility of this case is dependent on the amount of data available from the implement through ISOBUS. The sensor data related to field operations is available, but most of the system's internal data is not. It depends on the type of malfunction whether the available data is sufficient to diagnose the problem and give repair instructions to the user. The aggregating server should expose all of the sensor and internal data available from the underlying server for centralized remote access.

Data acquisition from a field machine
This use case refers to the logging of process data from selected data points. All data exposed by a UA server can be logged. (Piirainen, 2014). For this use case, the aggregating server must be able to subscribe to certain aggregated data nodes and store the values to a database for later retrieval.

Site-specific application using a variable-rate applicator
Piirainen suggested this use case as an alternative for site-specific application performed locally by a task controller. He concluded that this could possibly be achieved by directly writing to the variables exposed by the tractor's UA server, but that the real-time constraints involved in this are too much for this kind of control operation. He suggested an arrangement where the tractor server could be embedded with a client that could download the task data from another UA server, but that this was outside the scope of his thesis. If these clients were added, the aggregating server could be the server hosting this task data.

Updating execution plan and machine settings
This use case is also about writing data to the tractor server. A possible solution would be to use the same arrangement where a UA server would host the data which the tractor servers would then download. Some machine settings could possibly be changed by writing directly to the UA variables exposing them.

Monitoring and tracking machines in real time
This use case is dependent on the data exposed by the tractors UA server from the tractor network, the implement and the GNSS device. From the point of view of the aggregating server, the requirement for this use case is that it must expose all of the relevant data in its own address space and allow subscribing to them.

Backtracking field operations
The data requirements for this use case are the same as for the previous one, with the addition that the data should also be logged.

Logging of working hours and total area
For the aggregating server, the requirements for this use case is the logging of the data points from which the working hours and total area can be derived. The total area measurement is directly available as process data from the implement. In the previous thesis it was suggested that working hours could be logged by determining whether the tractor is moving from the GNSS data.

Benchmarking of machines and employees
This use case is about logging of certain data from the underlying servers and deriving performance indicators from that data. The required data contains the working width and speed of the implement for deriving the machine's area working capacity, momentary yield measurements for throughput capacity, GNSS data for determining whether the machine is currently working, and fuel consumption which is available from the SAE J1939 network (SAE, 2013).

Fault detection and diagnostics
This use case is about utilizing the data exposed by the tractor servers for predictive fault detection and fault diagnostics. In the previous thesis it was noted that predictive fault detection could in principle be based on the data exposed by the UA server from the SAE J1939 and ISOBUS networks, but that the required sample rate and amount of data to be transferred make this unfeasible.

Fault diagnostics on the other hand can be done with offline data. When a machine malfunctions or shows signs of an impending malfunction, the farmer or the maintenance service provider can remotely access the logged data and try to ascertain the cause for the malfunction. Whether this is successful is dependent on the data available from the tractor and implement networks that is exposed by the UA server. Again, from the aggregating server's point of view the requirements for this use case is the possibility to subscribe to and log the relevant data from the underlying server.

Combined seeding and fertilization execution with ISOBUS work-set
This is a use case defined by Ronkainen for the CLAFIS project. It involves a lot of steps, but the basic principle is that the FMIS is used to generate plans for seeding, fertilization and scheduling, after which a task file is generated from these plans. The potential role of the aggregating server in this use case is again hosting the task file which the tractor servers can then download.

4.3 Data requirements

The preliminary data requirements for the system are the tractor, implement, and GNSS data from the ISOBUS UA server and the means to transfer data between the aggregating server, the underlying servers, and UA clients.

The aggregating server must be able to gather relevant data from the underlying server and present it in a pre-defined structure in its own address space. Since the aggregating server is in the end intended as a centralized access point for multiple different kinds of farm equipment, which are likely to have different information models depending on the type of equipment, it should be able to expose the relevant data in its own address space using the same general information model regardless of the source information model (although in some cases a more varied representation on the aggregating server could be desirable). Depending on the desired structure of the aggregating address space, it may be possible to expose the relevant information using only the OPC UA standard types.

The aggregating server should also have a database for the storing of selected process data. This database must be able to store the data in a way that it can later be retrieved with a specified NodeId and time interval.

4.4 Other requirements

The most computationally intensive function of the aggregating server is the browsing of the underlying address spaces that are to be aggregated. However, it is likely that it doesn't need to be run often, since the connection between the nodes on the aggregating server and the corresponding nodes on an underlying server stays in place even if the underlying server is restarted. The only cases when it is needed to run is when a new device with its own server is added, or when the address space of an existing server changes so that it needs to be remapped. Nevertheless the time required shouldn't be distractingly long, especially when considering the relatively small address spaces of ISOBUS devices.

Depending on the structure of the target address space, the nodes from different underlying servers can be merged under the same object or folder in the aggregating address space. For this reason the aggregating server should have a way of identifying from which underlying server a node in the aggregating address space originated from.

5 Design

The primary focus point for the design of the aggregating server is the address space mapping from the underlying servers. For this reason, some of the general requirements for this type of aggregating server defined in section 4 are omitted from the prototype design. The final state of the prototype implementation is presented in more detail in section 6.

Section 5.1 gives a general description of the different components of the aggregating server. Section 5.2 describes the browsing algorithm and rule engine that constitute the address space mapping for the use case "Adding a new work machine to the aggregating server" defined in section 4.2.1. These parts of the design are explored in the most detail since this use case is the focus of this thesis. Section 5.3 describes the relaying of the services required for the use case "Reading and writing data through the aggregating server" defined in section 4.2.2 and also the Historical Access services required for the use case "Reading historical data from the aggregating server" defined in section 4.2.3.

5.1 Application overview

The relevant components of the software architecture of the aggregating server are presented in Figure 8.

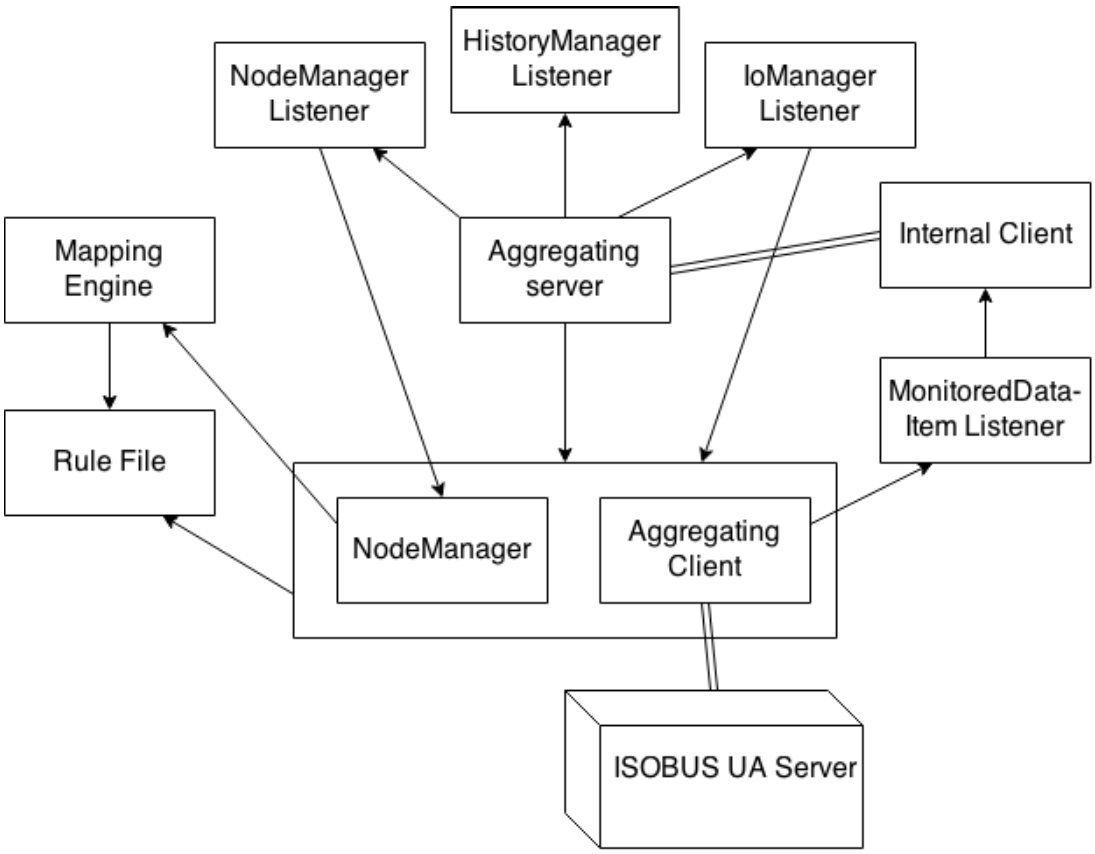


Figure 8. Architecture of the aggregating server application. The arrows denote that the source of the arrow uses the target of the arrow. The double lines denote a connection between a UA client and a UA server. The box surrounding the NodeManager and the Aggregating Client indicates that they come in pairs; for each underlying server there exists one NodeManager and one client.

The central piece of the architecture is the aggregating server itself, which exposes the address space and manages connections to the clients. It also acts as a user interface and manages the overall functionality of the application.

For each underlying UA server, the aggregating server instantiates a pair consisting of a NodeManager and a UA client. The aggregating clients are simple UA clients which the aggregating server uses to connect to each underlying server. They are used to browse through the address spaces of the underlying servers and search for nodes that are to be mapped to the aggregating address space. The browsing algorithm is presented in its entirety in section 5.2.1.

NodeManagers are responsible for the manipulation of the server address space, i.e. the creation and deletion of nodes. The aggregating server also has a root node manager and another internal node manager not shown in the figure. The former of these creates the default UA address space, consisting of the default folder structure, the type definitions and the server object. The latter one is used to extend the initial folder structure. The NodeManager which is shown in the same rectangle with the aggregating client is used to create the nodes in the address space of the aggregating server while a group of nodes returned by the aggregating client is evaluated by the mapping engine.

Each NodeManager has its own NamespaceIndex, and therefore all nodes in the aggregating address space originating from the same underlying server are exposed with the same unique NamespaceIndex. This is useful, for example, in a case where an underlying server is remapped and all the existing aggregating nodes for that specific server are deleted before creating new ones. The NamespaceUri for each underlying server in the aggregating address space is the ServerUri fetched from the ServerArray property of the underlying server. This same ServerUri also appears in the ServerArray property of the aggregating server, although with a different index number than the NamespaceIndex.

The Mapping Engine is used to evaluate the group of nodes returned to the NodeManager from an underlying server. It goes through the group of nodes given to it while referring to the rule file, and creates the aggregated nodes according to the rules. The design of the mapping engine is presented in more detail in section 5.2.2.

NodeManagerListener, MonitoredDataItemListener and the internal client are all used in handling subscriptions. A subscription made to a node in the aggregating server must be relayed to the corresponding node on the underlying server, and the changing values must be updated back to the aggregating node.

NodeManagerListener listens and reacts to changes made to the aggregating address space. When a subscription is made to an aggregating node, it looks up the correct aggregating client according to the NamespaceIndex of the node. It then uses the aggregating client to get the NodeId of the corresponding node on the underlying server, and sends this Id-pair to the MonitoredDataItemListener through the aggregating client.

Each aggregating client has its own MonitoredDataItemListener. MonitoredDataItemListener listens to data change events which are sent by the underlying server whenever the subscribed attribute of a particular node changes. It receives the NodeId from which the data change originated as well as the new value for the attribute. Using the originating NodeId it finds the corresponding NodeId in the aggregating address space, and uses the internal client to write this new value to the correct attribute of the aggregating node.

IoManagerListener listens to read and write calls in a similar fashion as the NodeManagerListener listens to subscriptions. When read or write is called to an attribute of a certain node, the IoManagerListener intercepts this service call, finds the corresponding aggregating client and the corresponding node from the underlying server, and uses the aggregating client to relay the service call to the underlying server. In the case of a read call the value from the underlying server is returned to the IoManagerListener, which then uses the aggregating NodeId to change the value of the aggregating node.

HistoryManagerListener listens to Historical Access services. It contains function templates that can be used to override these services. When the aggregating server is integrated with a historical database, these functions can be used to define the implementation of the services with respect to the database.

5.2 Address space mapping

The address space mapping of an underlying server is described in this section. Section 5.2.1 describes the algorithm that browses through the underlying address space searching for nodes that are to be mapped to the aggregating address space. The algorithm groups the appropriate nodes and sends them to the rule engine described in section 5.2.2. The engine evaluates these given nodes against a rule file and creates the corresponding nodes to the aggregating address space according to the rules.

5.2.1 Browsing algorithm

The operation of the mapping algorithm in a case where a new UA server is connected to the aggregating server is illustrated in Figures 9 and 10.

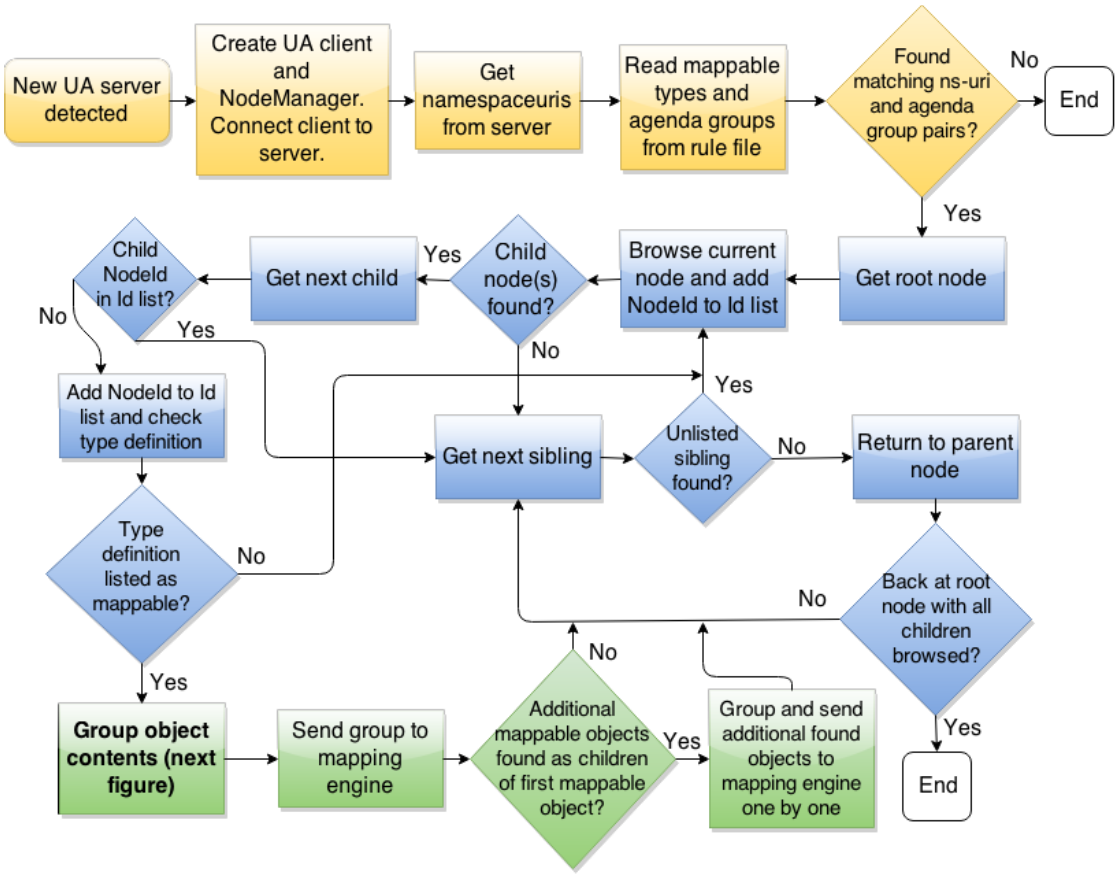


Figure 9. Flowchart depicting the operation of the mapping algorithm when a new UA server is connected. The bottom left box depicts a separate grouping algorithm which is initiated when a mappable object node is found. Its inner functionality is shown in Figure 10. The orange blocks represent the initialization phase, the blue blocks represent the browsing of the underlying address space, and the green blocks represent the object grouping and mapping phase of the algorithm.

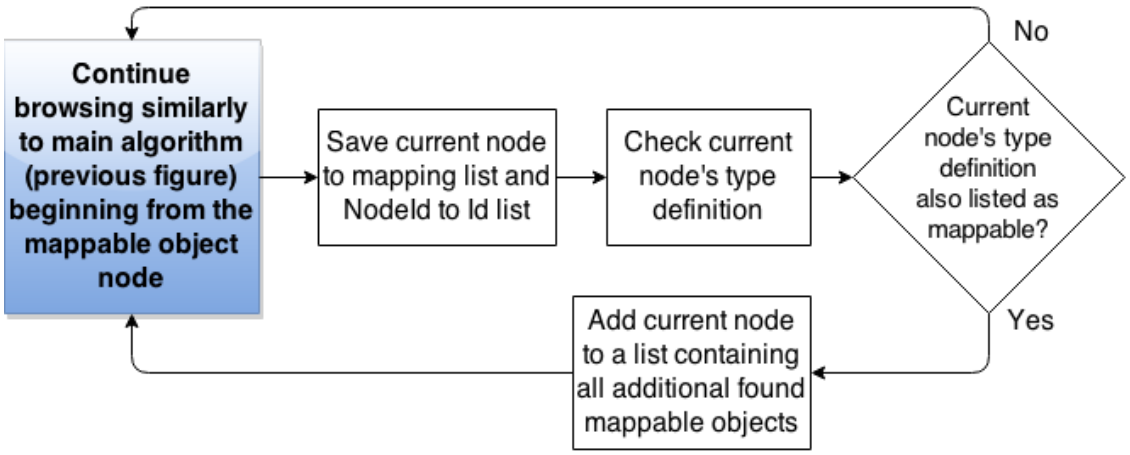


Figure 10. The grouping algorithm which collects all the contents of a mappable object for the mapping engine. The leftmost box means that the contents of the object node are browsed through in the same manner as the main algorithm depicted in Figure 9 browses through the whole address space (represented by the blue blocks). The differences are that the grouping algorithm only browses the nodes that are ultimately children of the mappable object, and that when the grouping algorithm encounters another mappable object inside the original mappable object, it is stored to a list containing all additional found mappable objects instead of initiating a new grouping algorithm instantly.

The orange blocks in Figure 9 represent the initialization phase of the browsing algorithm. The algorithm is initiated immediately after a new UA server is detected via the Discovery server or its address is entered manually. First, a UA client and a NodeManager is created for the underlying server. The client is connected to the server, after which it reads the NamespaceUris of the underlying server from its NamespaceArray property. When these are found, the server application checks what information models are supported by the mapping system from the beginning of the rule file. In the context of the rule file, these supported information models are called agenda groups. This term comes from the rule engine presented in section 6.1.1. It is used here to distinguish between the NamespaceUris found on the underlying server and the corresponding NamespaceUris that are used to group the rules. If no matching NamespaceUri-agenda group pairs are found, the algorithm ends here and no mapping is initiated. If one or more pairs are found, the aggregating client collects the NodeIds associated with these pairs from the rule file and begins browsing the underlying address space.

The blue blocks in Figure 9 form the main part of the browsing algorithm, representing the phase where the aggregating client actually browses through the underlying address space looking for mappable nodes. Browsing begins from the root node of the underlying server. It is implemented as a recursive depth-first search algorithm. Starting from the root node, each node in the address space is browsed. By default the Browse service returns all references originating from the target node. In the mapping algorithm these are limited to hierarchical and forward references. This allows browsing through all objects in the address space while minimizing unnecessary steps and preventing nodes from being skipped by the algorithm. The algorithm browses the first child of each node it encounters until a leaf node is reached, after which it returns to the previous node and browses the next child node if available. In this context a child node of a node is defined as a node which is connected to the parent node by a forward hierarchical reference originating from the parent node.

The algorithm keeps a list of all NodeIds that have already been browsed. Before browsing the next child node, it checks whether the NodeId of that node is already on the list. If it is, it means that this node has already been reached from some other reference and that there is no need to browse it again. The algorithm then traces back its steps to the first node that still has a child node whose NodeId is not yet on the list.

At each node the type definition of the current node is checked. If it does not match any of the mappable types read from the rule file the node is browsed and the algorithm continues normally. If a match is found, the operation of the main algorithm is suspended and a separate node grouping algorithm is initiated.

The grouping part of the algorithm is represented by the green blocks in Figure 9 and by Figure 10 in its entirety. Figure 10 is a zoom-in to the bottom left block of Figure 9. The grouping algorithm browses through the address space in the same manner as the main

algorithm. However, the root node for the grouping algorithm is the object node with the mappable type definition. Thus, the whole addres space for the grouping algorithm consists only of the contents of the mappable object. The operations of the grouping algorithm for each individual node are as follows:

1. **Check the NodeId of the node.** This is done identically to the main algorithm to prevent the same node from being browsed twice. If the NodeId is already on the list, the rest of the operations for the current node are skipped and the browsing continues

2. **Save the node to the mapping list.** Each node the grouping algorithm encounters for the first time are stored to a list. This list will ultimately contain the original mappable node which initiated the algorithm and all nodes belonging to it. When the grouping algorithm is finished, this list will be sent to the rule engine for evaluation and mapping.

3. **Check the type definition of the node.** The grouping algorithm compares the type definition of each node to the list of mappable types similarly to the main algorithm. It may be possible that a mappable object contains nodes that are also defined as mappable. If the type definition of the current node matches any of the ones listed as mappable, in addition to the mapping list in the previous step, it is also put on a list containing all additional found mappable objects.

When all nodes belonging to the mappable object are browsed through, the grouping algorithm is finished and the main algorithm continues. When the grouping algorithm is finished, the list containing the mappable object and its contents is sent to the mapping engine and the main algorithm is again suspended. At this point the mapping engine evaluates the mappable object along with its contents against the rules and creates the appropriate nodes to the aggregating address space. When this is done, the main algorithm is at a point where it has finished the two leftmost green boxes shown in Figure 9.

When the initial object is mapped, the main algorithm checks the list of additional found mappable objects created by the grouping algorithm (step three on the above list). If additional mappable objects were found, each of them are separately sent to the grouping algorithm, after which their contents are sent to the mapping engine. This means that the additional mappable objects are sub-objects of the original mappable object and that their contents are sent to the mapping engine a second time. The reasoning for this design comes from the fact that the mappable types are bound to a specific NamespaceUri. For example, there could be a general rule set concerning the standard FolderType objects, and another rule set for objects with a different NamespaceIndex that can be found inside object folders. If the additional objects would not be sent to the mapping engine again, only the rule set for the FolderType object would be evaluated for the whole group, and additional mappable objects with a different NamespaceIndex would be ignored. The handling of the additional mappable objects is illustrated in Figure 11.

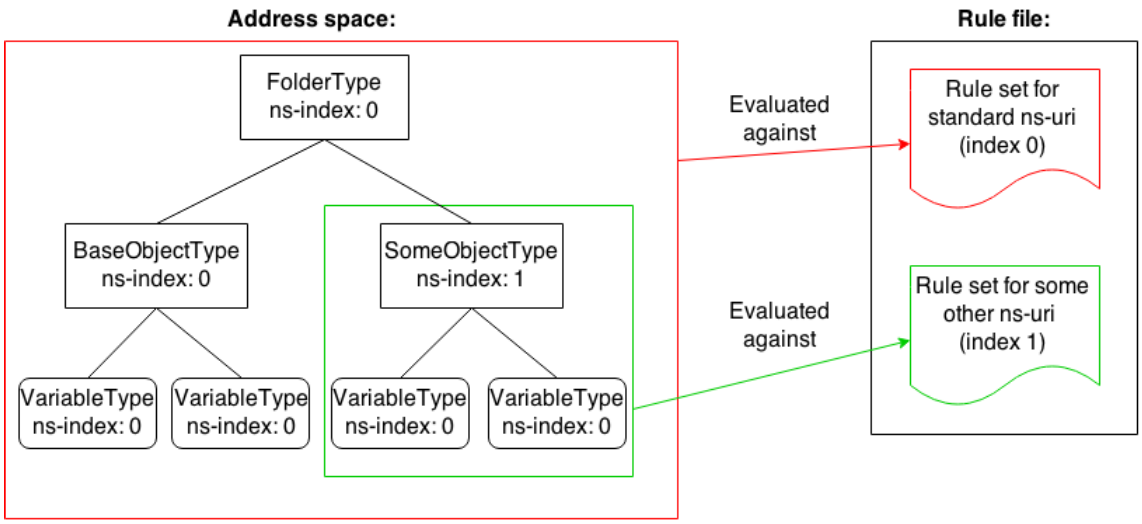


Figure 11. Example of a situation where a part of the original mappable object is sent to the mapping engine a second time.

In Figure 11, the main algorithm has two ObjectTypes it searches for; FolderType and SomeObjectType. First it encounters the uppermost folder object, groups all of its contents (surrounded by the red rectangle), and sends them to the mapping engine for evaluation. This group of nodes is only evaluated against the rules specified for the standard OPC UA types. When this is done, the algorithm checks the list produced from the first grouping algorithm run and notes that another object with a type that is defined as mappable with a different NamespaceUri was found. It then groups only the contents of this node (surrounded by the green rectangle) and sends them to the mapping engine, where they are evaluated against the rule set with a matching NamespaceUri (agenda group).

Because the first grouping algorithm run for the original mappable object already goes through every node ultimately originating from the object, no new mappable objects may be found during the subsequent runs. Thus the additional runs no longer check the type definitions for each node and only steps one and two from the above list are performed on each node. When all nodes from this additional list are also grouped and mapped, the main algorithm has completed the two rightmost green boxes in Figure 9 and continues browsing through the underlying address space.

The first grouping algorithm run uses the same NodeId list as the main algorithm. When the first object and all additional mappable objects have been dealt with by the mapping engine, the main algorithm continues normally and no longer browses to the mappable object or any of its contents. The algorithm continues from the next sibling of the mappable object and keeps searching for new mappable objects. The algorithm is finished once it has reverted back to the root node and there are no more child nodes left with their NodeId not on the list, indicating that all nodes have been browsed.

Sometimes it may be necessary to remap the address space of an underlying server that is already being aggregated. In such a case no new client or NodeManager is created. Instead, the internal client is used to collect and delete all the aggregating nodes and

references belonging to that specific underlying server from the aggregating address space. The correct nodes are identified according to their NamespaceIndex, which is unique for each underlying server. When the aggregating address space is cleared from the old nodes, the mapping algorithm is reinitiated using the original client already connected to the underlying server.

5.2.2 Rule evaluation and mapping

The operation of the mapping engine to which the node groups are sent by the algorithm is illustrated in Figure 12.

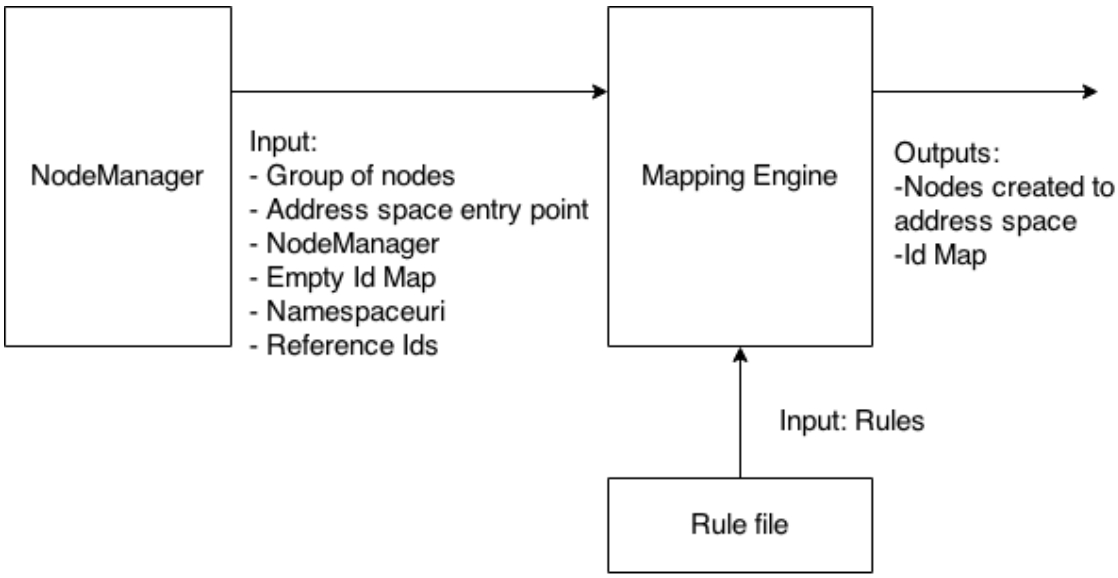


Figure 12. Inputs and outputs of the mapping engine

The required inputs to the mapping engine from the server application come through the NodeManager. These include:

The group of nodes to be mapped. These are the nodes that are grouped by the algorithm when browsing an underlying address space for node types that are defined as mappable. The group contains the mappable parent node and all its child nodes that can be found by following forward hierarchical references from the parent node. This means that each mappable object (containing all of its contents) in the underlying address space are mapped separately from each other.

Address space entry point. This is the NodeId for the node in the aggregating address space that the mapping engine uses as a starting point when creating new nodes. How this is chosen depends on the use case and address space structure of the aggregating server. This can be either a fixed node or it can be chosen or created according to some information found on the underlying server

NodeManager. This is the NodeManager created specifically to handle the

nodes for the underlying server that is currently being mapped. The mapping engine needs access to it to create the aggregating nodes continuously at the same time while evaluating the rules.

Id map. This is an empty data structure designed to contain NodeId pairs that connect an aggregating node to the corresponding node in the underlying address space.

NamespaceUri. The NamespaceUri for the information model to which the nodes currently being mapped belong to. This is used by the mapping engine to determine which set of rules it should apply for the current set of nodes.

Reference Ids. These are some of the reference NodeIds that are used in the address space structure of the object that is currently being mapped. These Ids are used in the conditional part of some of the rules to help determining the correct child nodes of the mappable parent node. Not all of the Ids used by the object are necessarily required as input for the mapping engine. Only the ones that are helpful in recognizing the nodes to be aggregated are needed. For example, one of the conditions of a rule could be that the node to be evaluated by the rule must be connected to another node with the HasComponent reference.

The mapping engine also takes the rule file as input. The rule file consists of a set of conditions and actions. A single rule consists of a name, its attributes, the conditional part, often referred to as the LHS (Left Hand Side) of a rule, and the consequence part, often referred to as the RHS (Right Hand Side) of a rule.

The name is merely an arbitrary string to uniquely identify a rule. The attributes of a rule specify additional conditions or properties for the rule that are not used in the LHS to evaluate the given node group. There are two relevant attributes for each rule in this design. The first one is the agenda group, which is used to group a set of rules for a specific case. One rule file can contain rules divided into multiple agenda groups, only one of which is evaluated at a time for a group of nodes. The agenda group is matched to the NamespaceUri given to the mapping engine. Thus, each agenda group represents an OPC UA information model. This way only the correct rules for each server are fired. If none of the NamespaceUris found on the server matches an agenda group in the rule file, no mapping is done.

Another attribute that is used is rule priority. Each rule in the same agenda group has a different priority indicator. A rule with a higher priority is evaluated for the complete set of nodes given to the mapping engine before any lower priority rules are fired. When the conditions of the highest priority rule are no longer met, the remaining nodes are evaluated against the conditions of the rule with the next highest priority. The rule flow for a group of nodes in a single agenda group with three rules is illustrated in Figure 13.

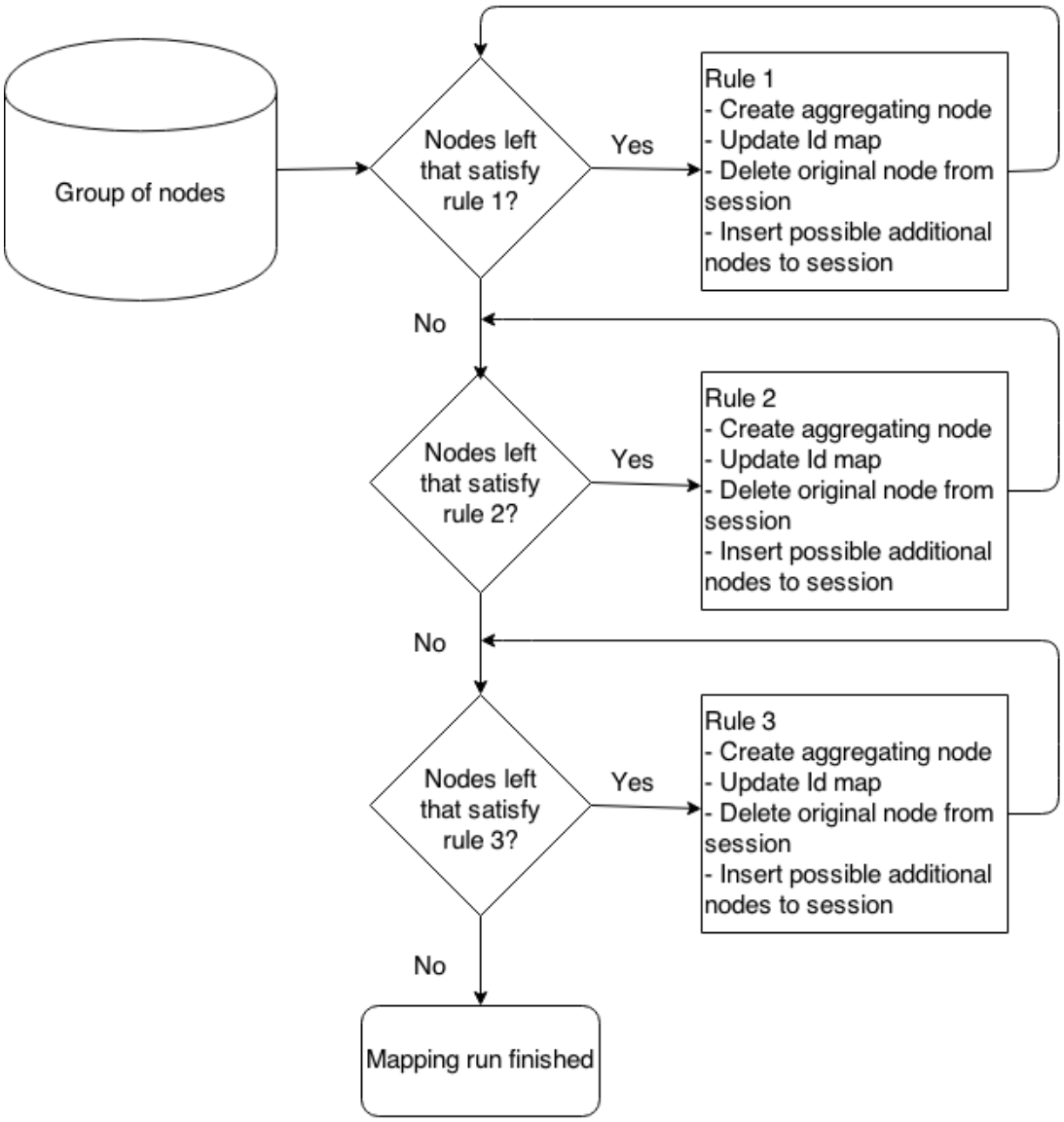


Figure 13. Rule flow for a single mapping run. The rules are numbered by order of priority, with rule 1 having the highest and rule 3 having the lowest priority.

The LHS contains the conditions and inputs for a single rule. An example of a condition is that a node with a specific type definition must be available for evaluation in the mapping engine. The LHS of a single rule can contain multiple different nodes as condition for the rule, or multiple conditions for a single node, such as the type definition, a specific parent node, and a specific reference originating or leading to it. When all the conditions for a rule are met, the nodes given as conditions are used as inputs for the RHS.

The RHS part reads the required attributes from the nodes given in the LHS and creates the corresponding aggregating node. Although a rule can take several nodes as input for condition evaluation purposes, only a single aggregating node is created per rule, although the engine is perfectly capable of creating multiple nodes from a single source node would that be required. Aggregating nodes are created by the mapping system during the evaluation of a rule and placed to the aggregating address space according to the

given entry point. When all required operations are done and the aggregating node has been created and added to the address space, both NodeIds from the original and the aggregating node are added to the Id map as a pair. Finally, the corresponding original node is deleted from the mapping system's memory so that it will not go through a rule again.

The RHS of a rule can also add new nodes to the engine's memory. This is used when a lower priority rule requires an input node created in a higher priority rule. For example, the first rule can be used to create a folder node, to which all later nodes are placed. This adds to the freedom of choice when determining the structure of the aggregating address space.

When there are no more nodes left to satisfy the conditions of any rule, the current mapping run is finished. At this point, all the aggregating nodes based on the underlying nodes have been created, and the connections between these nodes are stored to the Id map. This Id map is used by the server application to relay the service calls for the aggregating server to the underlying server.

In addition to the rules, the rule file also contains information specifying which types of nodes from which information models are to be mapped. A specific NodeId or the DisplayName of an ObjectType node is paired with the NamespaceUri it belongs to. These are read by the mapping algorithm in the beginning of the entire mapping procedure (fourth orange box from the left in Figure 9) after which it proceeds to browse the underlying server's address space while looking for nodes with some of the type definitions defined in the rule file.

5.3 Services

For the use case "Reading and writing data through the aggregating server", the Read, Write, and Subscribe services must be relayed to the underlying server. This is described in section 5.3.1. Section 5.3.2 describes the Historical Access services required for the use case "Reading historical data from the aggregating server".

5.3.1 Read, Write and Subscribe

The relaying of Read and Write calls is fairly straightforward with the IoManagerListener shown in Figure 8 on page 37. The procedure is illustrated in Figure 7 on page 32. The IoManagerListener listens to read and write calls directed at the aggregating server. Among other parameters, these calls contain the target node for the service call, the NodeId of said node, and the value to be returned or written.

The IoManagerListener contains a map of all the aggregating clients currently connected to an underlying server. These clients (and their corresponding NodeManagers) can be fetched from the map with the NamespaceIndex belonging to their NodeManager. When a Read call is directed at an aggregating node, the IoManagerListener intercepts this call. It first uses the NamespaceIndex of this node to check whether it corresponds

to some underlying node or not. If it does, it uses the NodeManager to get the Id map created by the mapping engine during rule evaluation for this specific server. Then from the Id map it gets the underlying NodeId that corresponds to the aggregating NodeId that was the target of the Read call. It then uses the aggregating client to create a new Read call directed at the corresponding underlying node. The result value from this read call is set as the value of the aggregating node and sent as the result value of the original Read call.

For Write calls this procedure is quite similar. The underlying server and the corresponding NodeId are fetched in the same manner as with the Read call. The correct aggregating client is then used to create a new write call targeted at the underlying node with the value from the original call as its parameter. If the operation is permitted, the given value is set as the value of the underlying node, after which this value change is also set to the aggregating node.

The relaying of subscriptions is not quite as straightforward. It is illustrated in Figure 14. It is handled with the help of the NodeManagerListener and the MonitoredDataItemListener both shown in Figure 8. When subscribe is called to one of the nodes in the aggregating address space, a Monitored Item for that node is created. The NodeManagerListener detects this, and finds the corresponding underlying server and NodeId in the same way as the IoManagerListener does for Read and Write calls. It sends the aggregating NodeId, the underlying NodeId and the AttributeId to which the subscription was targeted at to the correct aggregating client. This client then subscribes to the correct underlying node.

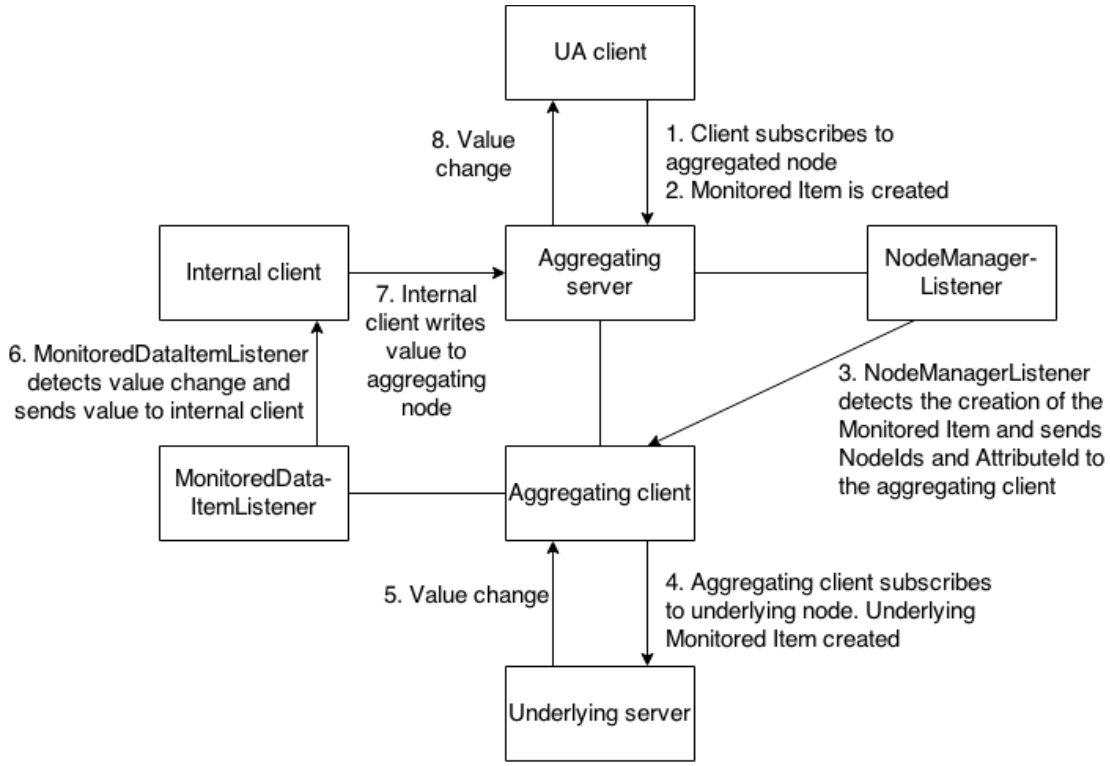


Figure 14. The relaying of subscriptions

When the aggregating client subscribes to an underlying node, a Monitored Item is created on the underlying server as well. The value changes from these underlying Monitored Items are listened to by the MonitoredDataItemListener. It has access to the same map containing the aggregating clients as the IoManagerListener does, and also to a list of NodeId pairs that contain the underlying NodeId and the corresponding aggregating NodeId for each Monitored Item. When a value change occurs in a Monitored Item, the MonitoredDataItemListener uses the internal client shown in Figure 8 to write this new value to the corresponding aggregating node with a Write call. This value change in the aggregating node is detected by the original subscription and returned to the client connected to the aggregating server.

5.3.2 Historical Access

Historical Access services are used to store data values or events and to read them later from memory or from a database. These services are listened to by the HistoryManagerListener shown in Figure 8. The HistoryManagerListener holds the server and database specific implementations for these services. Since the historical database is out of the scope of this thesis, these services are not implemented. If Historical Access and a database will be added in the future, these services will have to be implemented and tailored to the database that is chosen.

6 Evaluation

6.1 Prototype implementation

6.1.1 Mapping engine

The most fundamental decision regarding the aggregating server was choosing a rule syntax and engine that would be best suited for this kind of mapping application. One of the considered alternatives was the model proposed by Großman (2014). The main advantage would have been having the mapping rules included in the UA information models without an external rule file. However, requiring the underlying servers to have an information model extension for automated mapping was not considered viable for this thesis.

Out of the many available ontology mapping and rule engines, the one chosen for implementation was Drools. Drools is a production rule system based on an extended implementation of the Rete algorithm. As of release 6 (which is used in the prototype), Drools uses the PHREAK algorithm, which has evolved from the Rete algorithm but is no longer classified as a Rete implementation (JBoss Drools team, 2014). The advantages of PHREAK compared to Rete is that it allows for a bigger number of rules without significantly affecting performance. It also allows improved performance by the use of agenda groups and salience (JBoss Drools team, 2014), both of which are used in the rules for the prototype.

Drools was not the only choice for the rule engine, but it was one of the seemingly better choices and it seemed to fit well with the OPC UA Java SDK with which the prototype was being developed. Drools fulfills the requirements for the address space mapping system presented in section 5.2. It allows for the grouping and prioritizing of rules, it is flexible enough to permit many different kinds of address space transformations, like one-to-one, one-to-many, and many-to-one. This flexibility is increased by the fact that the consequence part of a Drools rule is written directly in code, removing the need for an additional rule parser. From a general view the main disadvantage of Drools is that it is Java dependent. If the server application is developed using some other language than Java, using Drools will require additional work. Possible options include implementing it as a service consumed by the code, or using Java Native Interface (JNI) to invoke Java from another language.

6.1.2 Server application

The aggregating server application was developed with the OPC UA Java SDK published by Prosys. The application architecture was illustrated in Figure 8 in section 5.1. Except for the mapping engine, rule file and ISOBUS UA server, all rectangles in the figure represent Java classes in the implementation. The internal client and aggregating clients are instances of the same class. The box surrounding the NodeManagers and aggregating clients represents a helper class containing one instance from both of these classes. The application consists of more classes than is shown in the figure, most of which are not relevant for this thesis. One exception is the DUaNode (Drools UaNode)

class, which is a helper class containing a node (represented by the UaNode class) and its type definition as a string. This is done to remove the need to pass the AddressSpace object to the rule engine for identifying node type definitions. DUaNodes are created by the algorithm for each node during the grouping phase.

The structure of the aggregating address space is given by WRM (Wapice Remote Management). It is layered into enterprises, sites, and assets. An enterprise can hold multiple sites, and one site can contain multiple assets. One work machine is considered an asset, and the farm to which it belongs is considered a site. These are all represented by folders in the address space. An asset folder contains the relevant information from a work machine as DataVariables, all of which have a few standard properties. No other hierarchy is defined inside an asset folder.

6.1.3 Requirements left out of the implementation

Historical Access was not implemented to the prototype, since the focus of this thesis was the mapping system and the integration of a database required for Historical Access would have significantly increased the workload of the thesis. The application has a MyHistorian class which contains function override template for the services related to Historical Access. Historical Access can be added by implementing these functions when a database is added to the application. The server also contains a function which initializes a working memory history for a NodeManager.

For the same reason as Historical Access, implementation of the automatic aggregation through a Discovery server was left out. Currently a server is added by entering its address manually to the server, after which everything else is handled automatically and the address space of the server is mapped and aggregated. Implementing discovery requires a dedicated Discovery server to which the underlying servers must register to, and a client in the aggregating server that is connected to the Discovery server and looks for underlying servers with the FindServers service.

The prototype implementation uses a fixed entry point to the address space, which is a site folder called “Vakola”. In the future, the site folder could be created according to some site information found on the underlying server. One option for this is to use an external service which would take the GNSS data from the work machine and use it to return the name of the location where the machine currently is. Likely a better option would be to add site information to a property in each underlying server.

Due to time restrictions, the implementation of persisting NodeIds was left out of the prototype. This means that if the aggregating server is restarted, the connections between the aggregating nodes and the proxy nodes are lost and all underlying servers need to be remapped again. For the scope of this prototype this does not matter, but nevertheless this is something that should be addressed in the future. If an underlying server is removed, the corresponding nodes in the aggregating address space can still be browsed, but service calls directed to their attributes will fail until the underlying server

becomes back online. In this case no remapping is required, assuming that the NodeIds of the underlying server persist.

6.1.4 Rules

The rules implemented for the prototype are used to transform a UA for ISOBUS address space to the WRM address space. A general representation of these rules is given below:

Rule 1: Map ISOBUSDevice
Salience > Rule 2
Input: Node of type ISOBUSDeviceType, Nodemanager
Actions:

1. Create new FolderType node with the name of the ISOBUSDevice node
2. Add new folder node to entry folder node with referenceType organizes
3. Copy possible required properties from ISOBUSDevice to new folder

Output: Insert created folder node to session, delete original device node from session

Rule 2: Map Device NAME
Rule 1 > Salience > Rule 3
Input: Node of type NAMEType, Nodemanager, FolderType node from rule 1
Actions:

1. Create new variable node with the name of the original NAME node
2. Copy attributes from original node
3. Add node to folder from rule 1 with referenceType organizes

Output: Delete original node from session

Rule 3: Map process data variables from all ISOBUSDeviceElements one by one
Salience < Rule 2
Input: Node which is organized by an object with displayname "ParameterSet", Nodemanager, folder inserted from rule 1, the "organizes"-reference between the node and "ParameterSet"
Actions:

1. Get the parent element of the input variable node
2. Get all grandparent elements of the input variable node
3. Get the EngineeringUnit and EURange properties from the original node
4. Parse WRM type display name for the new node from grandparent elements' display names and the engineering unit
5. Create new variableType node with the parsed display name
6. Add nodeIds from both old and new variables to ID-map
7. Copy attributes from old variable
8. Add the EngineeringUnit and EURange properties to new variable
9. Add new variable node to the folder node inserted from rule 1 (referenceType organizes)

Output: Delete original variable node from session

The inputs and outputs represent the information exchange between the rules during a single mapping session. The aggregating nodes are created under “Actions”. The actual rules implement these phases as Java code. The rule file also contains the required Java imports as well as the mappable NamespaceUri-type definition pairs. The rule file used by the prototype is presented in appendix 2. What these rules actually do in the address space is illustrated in appendix 3.

The mappable types are defined in the rule file as the NodeId of the mappable type definition. This worked for developing the prototype, since only a single server was used. However, the NamespaceIndices of a server are not fixed, so if a server has the NamespaceUri of the ISOBUS information model on a different index, a wrong or non-existing type is defined as mappable. This would be easily corrected by using the DisplayName of the mappable type instead of the NodeId. However, the implementation of the prototype ISOBUS UA server prevented this. Contrary to OPC UA specifications, some of the variables on the ISOBUS server lacked type definitions. This resulted in null pointer exceptions when the algorithm tried to get the DisplayName of each node it encountered.

6.2 Objectives and setup

The setup used in the experimentation consists of the prototype aggregating server, the UA for ISOBUS server, one Junkkari Maestro ECU, 22 DDOP files emulating additional implements, and UaExpert, a graphical UA client software. 15 of the DDOP files were created by hand but according to various real machines. 6 files were different versions of the Junkkari Maestro DDOP (3.59, 3.61, 3000 de, 4000 de, 4000 en, 4000 lux en). One file was from a Juko 200 seed drill.

The objective of the experimentation is to confirm that the mapping system is capable of successfully aggregating implements with differing UA address spaces. In addition to the mapping algorithm, this also evaluates that the prototype implementation of the ISOBUS UA server is capable of instantiating different implements in its address space. The DDOP XML files were converted to ISOBUS messages using an XML to binary parser created specifically for this thesis.

After the implements have been aggregated, UaExpert is used to test that the read, write and subscribe services are relayed correctly and work as intended through the aggregating server.

6.3 Results

The address spaces instantiated from the Junkkari Maestro ECU as well as from all of the DDOP files converted to ISOBUS messages were mapped and aggregated success-

fully. The services tested with UaExpert also worked as expected. Debug points in the code were used to ensure that the correct values were returned by read and subscribe.

During testing, the aggregating server was run on a desktop computer with an Intel Core 2 Duo 3.06 GHz CPU and 4 gigabytes of RAM. The ISOBUS UA server was running on a laptop with an Intel Core i5-2520M 2.50 GHz CPU and 4 gigabytes of RAM. The operating system on both computers was Windows 7. The mapping procedure for the implements was reasonably fast, ranging from 5 to 20 seconds. However, both the ISOBUS UA server and the aggregating server were on the same local area network situated in Espoo during testing. When the mapping algorithm was tested with the WRM server situated in Tampere, the address space mapping took considerably longer than could have been expected from the size differences of the address spaces. The main bottleneck for the algorithm seems to be the relaying of separate Browse calls for each node in the underlying address space. This could possibly be circumvented by using Query instead of Browse, but this was not possible since the ISOBUS UA server did not support this service. However, it should be noted that since the WRM server does not support the UA for ISOBUS information model, a different set of rules were used which might have also affected the execution time.

Examples of address spaces before and after the mapping procedure are presented in Figures 15 and 16.

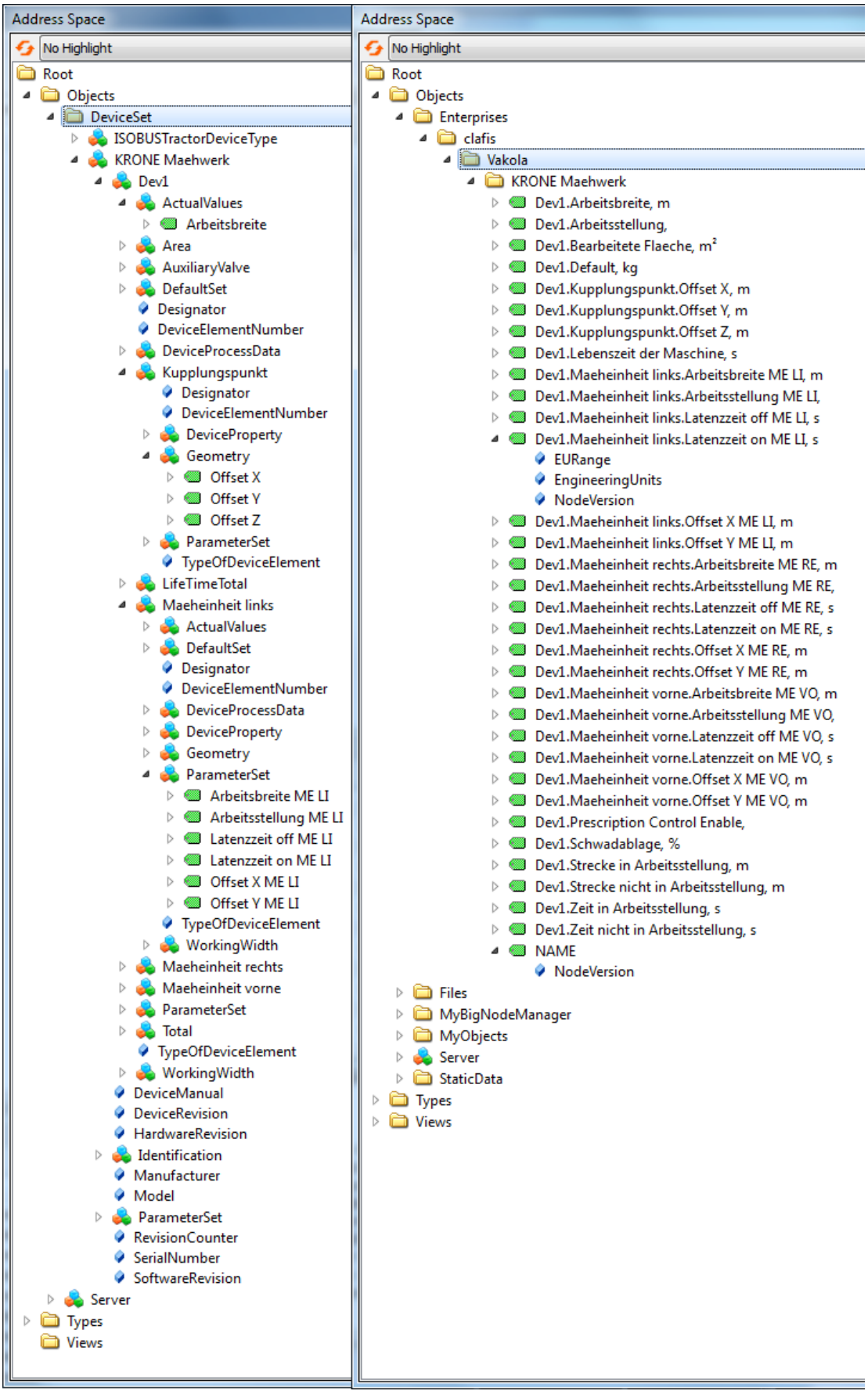


Figure 15. Original and aggregated address spaces for a mower implement by Krone.

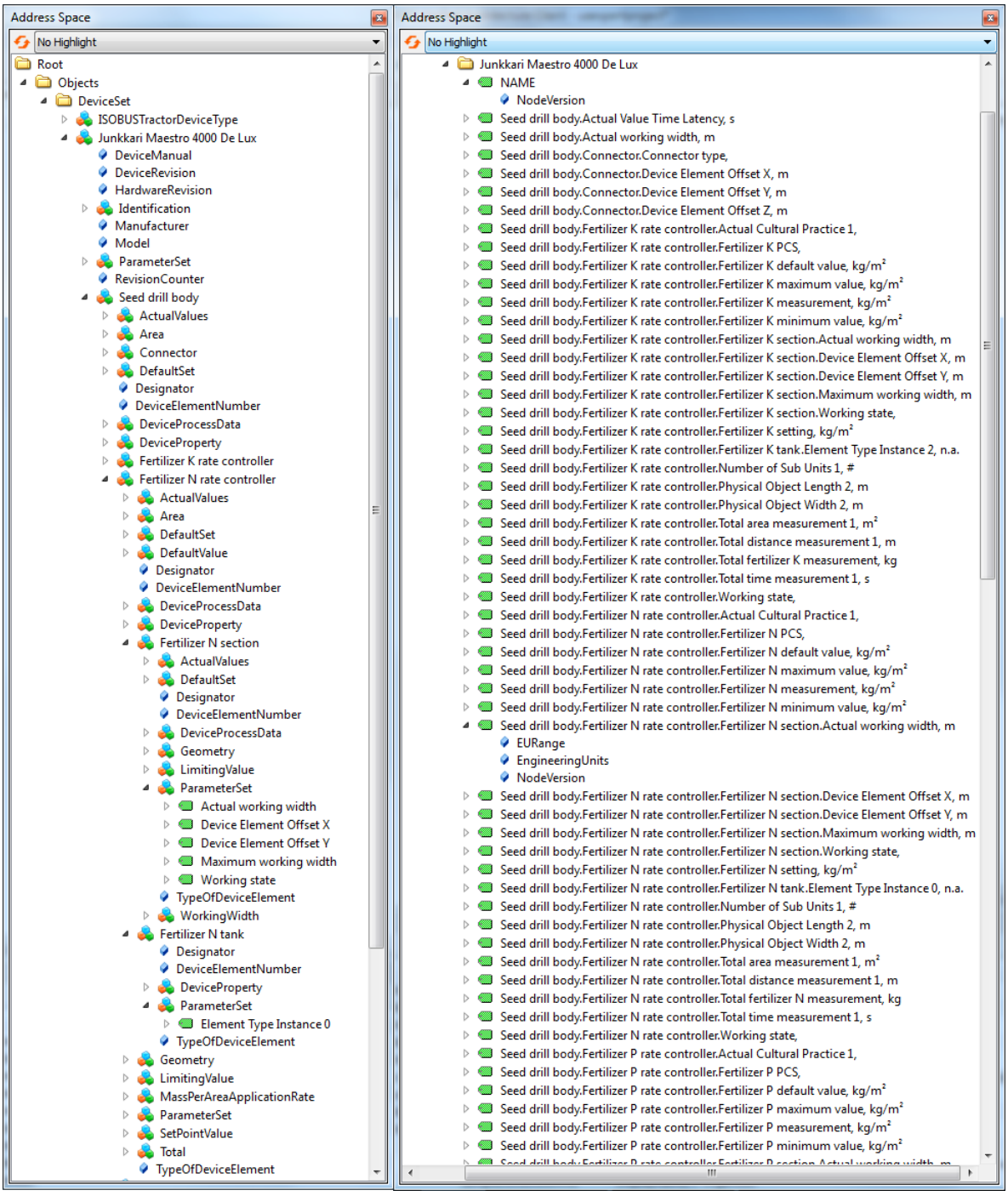


Figure 16. Original and aggregated address spaces for a Junkkari Maestro 4000 lux seed drill.

7 Conclusions

7.1 Summary and conclusions

The goal of this thesis was to evaluate OPC UA and aggregating servers as a possible technology for remote access to and data acquisition from agricultural work machines utilizing the ISOBUS communication protocol. Based on some existing requirements and some new ones defined in this thesis, a design for an aggregating server for this purpose was created. Based on this design, a partial prototype aggregating server was implemented.

The main use case for the prototype was the automatic mapping and aggregation of various work machines running UA servers with differing address spaces. For this, a mapping algorithm was created based on the Drools rule engine. By defining three rules, the relevant data from the 23 tested implement address spaces was successfully mapped to the aggregating address space following a structure defined by WRM. Data transfer between the underlying server and the aggregating server was implemented by relaying service calls directed at the aggregating server to the corresponding underlying server.

The requirements for the aggregating server were defined in chapter 4. The basis for the requirements definition were the needs of the various stakeholders involved with a system the aggregating server could be a part of. One of the requirements was to ensure that the aggregating server maintains the functionality and fulfills the requirements set for the ISOBUS UA server. This required the exposure of all relevant data nodes from the ISOBUS UA servers on the aggregating server, and the relaying of certain required services between these nodes. For centralized monitoring and data acquisition, the relevant data from each underlying server needed to be accessible through the aggregating server. For storing values from specific data points, the aggregating server needs to be integrated with a database. In some cases, writing data to specific nodes on specific machines was also required. To assist in this, the aggregating server was required to present the relevant nodes in a compressed manner while disposing of the irrelevant nodes. Because the machines connected to the aggregating server can change or increase in number, the gathering and mapping of the relevant nodes needs to be done at least semi-automatically.

The design of the aggregating server based on the requirements was defined in chapter 5. For the automatic address space transformation and mapping, a rule file was defined against the OPC UA information model created for ISOBUS work machines. This approach was found to be sufficient in mapping the relevant nodes from different work machines. By defining additional rules, it is also extensible to the address space aggregation of other OPC UA suitable equipment with different information models. The relaying of services was enabled by pairing the NodeIds of the aggregating nodes with their counterparts on the underlying servers. Database integration was left out of the design.

The evaluation of the design by implementing and testing a prototype was presented in chapter 6. The purpose was to ensure that the mapping algorithm and the defined rules were capable of mapping the address spaces from different work machines to the aggregating address space, and that the required services were successfully relayed between these nodes. All of the tested address spaces were successfully mapped, and services were successfully relayed to the underlying server and back. Evaluation of the prototype on different networks revealed some performance issues with the browsing part of the mapping algorithm. On the local area network where the prototype was implemented and tested with the ISOBUS UA server, no significant delays were observed. However, on a wireless network, sending the Browse calls for each underlying node took considerably longer.

OPC UA is one of the most promising current technologies for interoperability and integration. The strength and flexibility of OPC UA comes from the ability to define less and more general types to represent devices from different domains. Since ISOBUS is a common and steady standard in agricultural machinery and OPC UA is highly scalable, adopting OPC UA early can ease the future expansion of FMIS data collection significantly. OPC UA information models could also be defined for different kinds of equipment utilized in agriculture. For these information models, rules could be defined to allow the automatic aggregation of future devices of the same type. The benefits of OPC UA seem to be emphasized more as the scale of the system it is implemented on grows. In conclusion, OPC UA aggregating servers seem to be a promising technology for remote access to agricultural work machines.

7.2 Future work

There are a few suggestion how future work could improve the design presented in this thesis. One is optimizing the browsing algorithm, since the actual environment of the aggregating server probably utilizes a 3G network or some other not so fast wireless network. The performance of the algorithm could be improved by reducing the portion of the underlying address space for which Browse calls are sent. Currently, one Browse call is sent per each node in the entire address space. However, the current design only maps the nodes in the Objects folder, so the browsing of the address space could be restricted to this folder. Also, sending Browse calls to Property nodes is unnecessary, since they are always leaf nodes. Another option that could be researched is replacing the use of the Browse service with the Query service. This could reduce the amount of service calls significantly, but it is not certain whether it can be used to reliably retrieve all the required contents from an unknown address space.

Another suggestion for future convenience is creating some sort of interface for designing the rules. The rules are currently written directly in code, and a new set of rules is required whenever a new type of device with a different information model is added to the repertoire of the aggregating server. Implementing and interface through which the

user could define simple mapping rules which would then be generated as code in the rule file is likely less prone to errors than writing the rules directly in code.

Possible future work also includes implementing the requirements omitted from the current design. These include implementing the Historical Access services and database integration, as well as hosting files in the aggregating server which the underlying servers could then download.

References

Asikainen, J., “OPC UA Java history gateway with inherent database integration”, Master’s Thesis, Aalto University School of Electrical Engineering, Espoo, Finland, 2013.

Bliss, M., “Farm management”, 2015, *Encyclopædia Britannica Online*. Retrieved 24 April, 2015, from http://global.britannica.com/EBchecked/topic/201926/farm-management

Grisso, R., Alley, M, Holshouser, D., Thomason, W., “Precision farming tools: soil electrical conductivity”, Virginia Cooperative Extension, Publication 442-508, 2009.

Großman, D., Bregulla, M., Banerjee, S., Schulz, D., ”OPC UA server aggregation – the foundation for an internet of portals”, Emerging Technology and Factory Automation (ETFA), IEEE, 2014.

Hevner, A.R., March, S.T., Park, J., Ram, S., “Design science in information systems research”, MIS Quarterly vol. 28 No. 1, pp. 75-105, 2004.

ISO, “Tractors and machinery for agriculture and forestry - serial control and communications data network - part 1: General standard for mobile data communication”, ISO 11783-1:2006(E), International Organization for Standardization, Geneva, Switzerland, 2006.

ISO, “Tractors and machinery for agriculture and forestry - serial control and communications data network - part 3: Data link layer”, ISO 11783-3:1998(E), International Organization for Standardization, Geneva, Switzerland, 1998.

ISO, “Tractors and machinery for agriculture and forestry - serial control and communications data network - part 10: Task controller and management information system data interchange”, ISO 11783-10:2009(E), International Organization for Standardization, Geneva, Switzerland, 2009.

JBoss Drools team, “Drools documentation version 6.1.0 Final”, retrieved 11 may 2015, from https://docs.jboss.org/drools/release/6.1.0.Final/drools-docs/html_single/

Mahnke, W., Leitner, S-H., Damm, M., “OPC Unified Architecture”, Springer, 2009.

Mondal, P., Tewari, V.K., “Present status of precision farming: a review”, International Journal of Agricultural Research 2 (1), pp. 1-10, 2007.

Mondal, P., Tewari, V.K., Rao, P.N, Verma, R.B., Basu, M., "Scope of precision agriculture in India", Proceedings of international conference on emerging technologies in agricultural and food engineering, Kharagpur, India. PMS 101/6, pp. 103, 2004.

Murakami, E., Saraiva, A.M., Ribeiro Jr., L.C.M., Cugnasca, C.E., Hirakawa, A.R., Correa, P.L.P., "An infrastructure for the development of distributed service-oriented information systems for precision agriculture", Computers and Electronics in Agriculture 58, pp. 37–48, Elsevier, 2007.

Nikkilä, R., Seilonen, I., Koskinen, K., "Software architecture for farm management information systems in precision agriculture"., Computers and Electronics in Agriculture 70, pp. 328-336, Elsevier, 2010.

O'Brien, J., Management Information Systems – Managing Information Technology in the Internetworked Enterprise. Boston: Irwin McGraw-Hill. ISBN 0-07-112373-3, 1999.

OPC Foundation, "OPC UA Specifcation Part 3: Address Space Model", OPC UA Specifcation Release 1.02, 2012.

OPC Foundation, "OPC UA Specifcation Part 4: Services", OPC UA Specication Release 1.02, 2012.

OPC Foundation, "OPC UA Specifcation Part 5: Information Model", OPC UA Specifcation Release 1.02, 2012.

OPC Foundation, "OPC UA Specifcation Part 8: Data Access", OPC UA Specifcation Release 1.02, 2012.

OPC Foundation, "OPC UA Specifcation Part 9: Alarms & Conditions", OPC UA Specifcation Release 1.02, 2012.

OPC Foundation, "OPC UA Specifcation Part 10: Programs", OPC UA Specifcation Release 1.02, 2012.

OPC Foundation, "OPC UA Specifcation Part 11: Historical Access", OPC UA Specifcation Release 1.02, 2012

OPC Foundation, "OPC UA Specifcation Part 13: Aggregates", OPC UA Specifcation Release 1.02, 2012

OPC Foundation, "OPC UA for Analyser Devices Companion Specifcation", OPC UA Companion Specifcation Release 1.1, 2013b.

OPC Foundation, “OPC UA for ISA-95 Common Object Model”, OPC UA Common Object Model Release 1.01, 2013.

OPC Foundation, “MTConnect OPC UA Companion Specifcation”, OPC UA Companion Specifcation Release 1.02, 2013a.

OPC Foundation, “OPC UA for Devices Companion Specifcation”, OPC UA Companion Specifcation Release 1.01, 2013c.

Peffers, K., Tuunanen, T., Gengler, C.E., Rossi, M., Hui, W., Virtanen, V., Bragge, J., “The design science research process: A model for producing and presenting information systems research”, Proceedings of the 1st International Conference on Design Science Research in Information Systems and Technology (DESRIST), 2006.

Piirainen, P., “OPC UA based remote access to agricultural field machines”, Master’s Thesis, Aalto University School of Electrical Engineering, Espoo, Finland, 2014.

Ronkainen, A., Nikander, J., Pesonen, L., Koistinen, M., Suomi, P., ”Use-case Description and Specification for System Functionalities”, FP7 – CLAFIS, Deliverable no. D1.1 – Version 1.0, 2014.

SAE, “Surface Vehicle Recommended Practice, Vehicle Application Layer”, SAE International, 2013.

Seilonen, I., Terho, S., Salmenperä, M., ”Is OPC UA technology for mobile work machines?”, FIMA-report, 2013.

Sørensen, C.G., Bochtis, D.D., ”Conceptual model of fleet management in agriculture”, Biosystems Engineering vol 105, Issue 1, pp. 41-50, Elsevier, 2010.

Sørensen, C.G., Fountas, S., Nash, E., Pesonen, L., Bochtis, D., Pedersen, S.M., Basso, B., Blackmore, S.B., ”Conceptual model of a future farm management information system”, Computers and Electronics in Agriculture 72, pp. 37-47, Elsevier, 2010a.

Sørensen, C.G., Pesonen, L., Fountas, S., Suomi, P., Bochtis, D., Bildsøe, P., Pedersen, S.M., ”A user-centric approach for information modelling in arable farming”, Computers and Electronics in Agriculture 73, pp. 44-55, Elsevier, 2010b.

Sørensen, C.G., Pesonen, L., Bochtis, D.D., Vougioukas, S.G., Suomi, P., “Functional requirements for a future farm management information system”, Computers and Electronics in Agriculture 76, pp. 266-276, Elsevier, 2011.

Appendices

Appendix 1. Data requirements for a fertilizing case, 2 pages
Appendix 2. Rule file, 2 pages
Appendix 3. Address space transformation, 1 page

Appendix 1. Data requirements for a fertilizing case

A heavily trimmed table of data requirements for a fertilizing case presented by Sørensen (2011)

Entity	Definition	Attributes/data	Availability of data	Future requirements
Actual and forecasted weather	Current weather and short term forecast	Type of weather parameter (temperature, humidity etc.) and its value	Current weather on field/parcel level, forecast on regional level	Forecast on field/parcel level, on-board measurement of current weather
Available machinery	Description of currently available machinery	List of available machines (type, ID, DCD file)	Available by manual creation	Automated downloading of DCD file to the FMIS
Updated moisture conditions	Description of current moisture information in the field	Field ID, soil moisture	Soil sensor network for data transfer, local weather station network for data transfer	Automated measurements and decision support function for decision making, automated moisture prediction models
Selected machine	Technical information of selected machine	Machine ID, DCD	Available by manual creation	Automated downloading of DCD file to the FMIS
Monitoring information	Monitoring data from the external sensors	Type of operational parameter, value of operational parameter, logging frequency	automatically generated	
Process information	Control information from internal sensors measuring work unit's perfor-	Type of operational parameter, value of operational parameter, logging frequency	automatically generated	

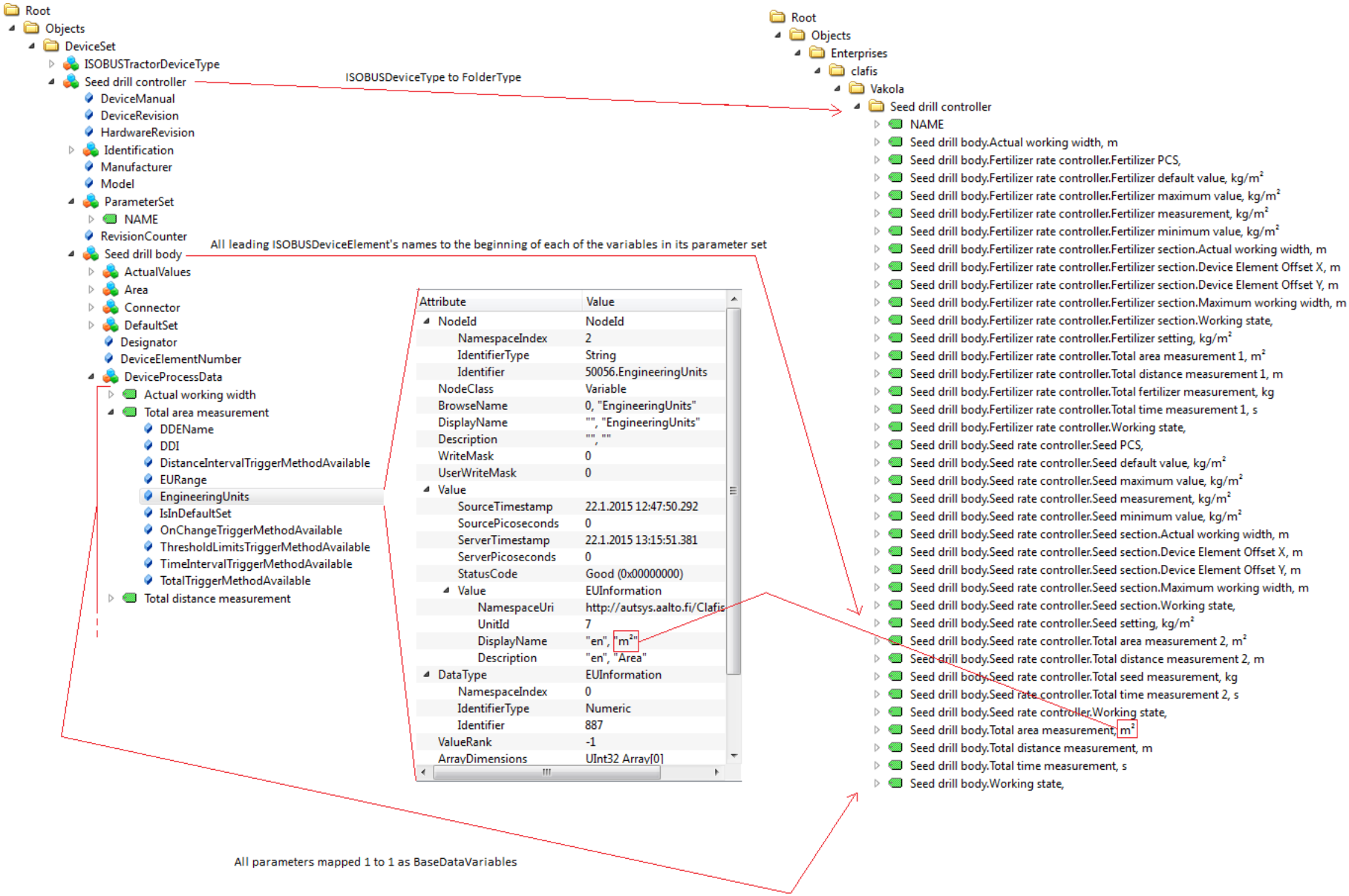
	mance			
Operation status and documented execution data	Current operation status	Machine ID, aggregated process information, current realized work performance compared to planned capacity (ha/h), current operation progress compared to schedule, remaining fertilizing work, recorded monitoring information (soil moisture, biomass values, etc.)	Automatically generated	
Overall task monitoring	Real-time status information and adjustments shown in the virtual terminal	Field ID, date and time, worker ID, machine ID, aggregated process information, current realized work performance compared to planned capacity (ha/h), current operation progress compared to schedule, remaining fertilizing work, recorded monitoring information, fertilizing process adjustments	Automatically generated	
Documented execution data	description of summary data for execution	Field ID, date and time, worker ID, machine ID, aggregated process information, summarized realized work performance (ha/h), summarized amount of applied fertilizers, summarized operation progress compared to schedule, remaining fertilizing work, summarized recorded monitoring information, notes for fertilizing process adjustments	Automatically generated	

Appendix 2. Rule file

```
//Mappable types:
//2,1002,http://autsys.aalto.fi/ClafisFinalProto/
//End of mappable types

//created on: 11.2.2015
package com.prosysopc.ua.samples.server

import com.prosysopc.ua.nodes.UaNode;
... rest of the imports hidden



//declare any global variables here
global com.prosysopc.ua.nodes.UaNode entryNode;
global org.opcfoundation.ua.builtintypes.NodeId hasComponentId;
global org.opcfoundation.ua.builtintypes.NodeId organizesId;
global org.opcfoundation.ua.builtintypes.NodeId hasTypeDefId;
global java.util.HashMap IdMap;


rule "WRM 1: Map device as an asset folder using DUaNode"
salience 11
agenda-group "http://autsys.aalto.fi/ClafisFinalProto/"
    when
        n : DUaNode(typeDef.equals("ISOBUSDeviceType"), dispName : node.getDisplayName())
        nm : MyNodeManager()
    then
        NodeId newId = new NodeId(nm.getNamespaceIndex(), UUID.randomUUID());
        FolderTypeNode mappedNode = nm.createInstance(FolderTypeNode.class, dispName.getText(),
        newId);
        nm.addNodeAndReference(entryNode, mappedNode, Identifiers.Organizes);
        insert(mappedNode);
        delete(n);
end

rule "WRM 2: Map device NAME"
salience 9
agenda-group "http://autsys.aalto.fi/ClafisFinalProto/"
    when
        n : DUaNode(typeDef.equals("NAMEType"), dispName : node.getDisplayName())
        af : FolderTypeNode();
        nm : MyNodeManager()
    then
        NodeId newId = new NodeId(nm.getNamespaceIndex(), UUID.randomUUID());
        UaVariable mappedNode = new CacheVariable(nm, newId, new
        QualifiedName(nm.getNamespaceIndex(),n.getBrowseName().getName()), n.getDisplayName());
        mappedNode.setAttributes(n.getAttributes());
        mappedNode.getValue().setSourceTimestamp(((UaVariable)
        n.getUaNode()).getValue().getSourceTimestamp());
        ((BaseNode) mappedNode).initNodeVersion();
        nm.addNodeAndReference(af, mappedNode, Identifiers.Organizes);
        delete (n);
end


rule "WRM 3: Map process data / parameter set contents one by one"
salience 8
agenda-group "http://autsys.aalto.fi/ClafisFinalProto/"
    when
        n : DUaNode()
        af : FolderTypeNode()
        nm : MyNodeManager()
```

```
        orgref : UaReference(
        this.getSourceNode().getDisplayName().getText().equals("ParameterSet")) from
        n.getReferences(hasComponentId, true)
    then
        UaNode parentElement = orgref.getSourceNode();

        //Parse grandparents displayname
        UaNode grandParentElement = parentElement.getReference(hasComponentId,
        true).getSourceNode();
        String parentElementDisplayName = grandParentElement.getDisplayName().getText();

        //Parse displayname from additional parent elements
        UaNode grandParentElementN = grandParentElement.getReference(hasComponentId,
        true).getSourceNode();
        while (grandParentElementN.getReference(hasTypeDefId,false).getTargetNode(
        ).getDisplayName().getText().equals("ISOBUSDeviceElementType")) {
             parentElementDisplayName = grandParentElementN.getDisplayName().getText() + "." +
             parentElementDisplayName;
             grandParentElementN = grandParentElementN.getReference(hasComponentId,
             true).getSourceNode();
        }

        //parse engineering unit
        QualifiedName engUnitName = new QualifiedName(0, "EngineeringUnits");
        UaProperty engUnitProperty = n.getProperty(engUnitName);
        String engUnit = "";
        if (engUnitProperty != null) {
             DataValue testi = engUnitProperty.getValue();
             int alku = testi.toString().indexOf("(en)");
             int loppu = testi.toString().indexOf("\n", alku);
             engUnit = testi.toString().substring(alku+5, loppu);
        }

        NodeId newId = new NodeId(nm.getNamespaceIndex(), UUID.randomUUID());
        IdMap.put(newId, n.getNodeId());
        LocalizedText displayName = new LocalizedText((parentElementDisplayName + "." +
        n.getDisplayName().getText() + ", " + engUnit), Locale.ENGLISH);
        UaVariable mappedNode = new CacheVariable(nm, newId, new
        QualifiedName(nm.getNamespaceIndex(),n.getBrowseName().getName()), displayName);
        mappedNode.setAttributes(n.getAttributes());

        //Copy required properties for the new variable.
        QualifiedName EURangeName = new QualifiedName(0, "EURange");
        UaProperty EURangeProperty = n.getProperty(EURangeName);
        if (EURangeProperty != null) {
             NodeId EURpropertyId = new NodeId(nm.getNamespaceIndex(),UUID.randomUUID());
             UaProperty newEURProperty = new CacheProperty(nm, EURpropertyId, new
             QualifiedName(nm.getNamespaceIndex(),EURangeProperty.getBrowseName().getName()),
             EURangeProperty.getDisplayName());
             newEURProperty.setAttributes(EURangeProperty.getAttributes());
             nm.addNodeAndReference(mappedNode, newEURProperty, Identifiers.HasProperty);
        }

        if (engUnitProperty != null) {
             NodeId EUpropertyId = new NodeId(nm.getNamespaceIndex(),UUID.randomUUID());
             UaProperty newEUProperty = new CacheProperty(nm, EUpropertyId, new
             QualifiedName(nm.getNamespaceIndex(),engUnitProperty.getBrowseName().getName()),
             engUnitProperty.getDisplayName());
             newEUProperty.setAttributes(engUnitProperty.getAttributes());
             nm.addNodeAndReference(mappedNode, newEUProperty, Identifiers.HasProperty);
        }
        ((BaseNode) mappedNode).initNodeVersion();
        nm.addNodeAndReference(af, mappedNode, Identifiers.Organizes);
        delete(n);
end
```

Appendix 3. Address space transformation



Address space transformation. The names of all leading device elements for each variable are put on the beginning of their name in the same order as they appear in the original address space. Seed drill body is the main element which contains all the other device elements.