Aalto University
School of Electrical Engineering
Department of Electrical Engineering and Automation

Ian L. Tuomi

Aggregating OPC UA Server for Flexible Manufacturing Systems

Master's Thesis
Espoo, July 7, 2015

Supervisor:	Ilkka Seilonen, D.Sc. (Tech.)
Advisor:	Ilkka Seilonen, D.Sc. (Tech.)

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical Engineering
Department of Electrical Engineering and Automation

ABSTRACT OF
MASTER'S THESIS

Author:	Ian L. Tuomi		
Title:	Aggregating OPC UA Server for Flexible Manufacturing Systems		
Date:	July 7, 2015	**Pages:**	viii + 53
Major:	Information and Computer Systems in Automation	**Code:**	AS-116
Supervisor:	Ilkka Seilonen, D.Sc. (Tech.)		
Advisor:	Ilkka Seilonen, D.Sc. (Tech.)		

Flexible manufacturing systems are becoming increasingly sophisticated as their efficiency is increased with additional software features. More of the information generated by the flexible manufacturing system is being gathered and stored, and increasingly utilized by higher-level systems in new ways to increase cost-effectiveness. The efficient flow of information from a flexible manufacturing system to higher-level information systems has therefore gained importance. Additionally, integrating the devices in the flexible manufacturing system is challenging due to the many communication protocols in use. A communication protocol named OPC UA and an aggregating server architecture are a possible way to modernize flexible manufacturing systems and solve integration problems they face. The goal of this thesis was to clarify the requirements and design of an aggregating OPC UA server as part of a FMS.

In this work, the requirements definition, design, and implementation of an aggregating server for flexible manufacturing systems were created, and the functionality demonstrated. The requirements of an aggregating OPC UA server for FMS were formed based on previous work. A software design fulfilling the requirements was detailed. A prototype based on the design was implemented and experimented on by developing example applications using it, complying with selected use cases.

External client access to devices, historization of data, alarms with user-defined alarm conditions, and mapping node values to custom functionality were identified as relevant requirements of an OPC UA aggregating server for FMS. The implementation of an aggregating server with these features using OPC UA was found to be practical. This thesis found no objections to the use of OPC UA in a FMS setting. The benefits of using OPC UA increase greatly when compatible equipment is available and does not need to be specifically integrated. However, the extent to which OPC UA is useful in communications between separate information systems is not yet known.

Keywords: FMS, OPC Unified Architecture, aggregating server

Language: English

Aalto-yliopisto
Sähkötekniikan
korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu
Automaation ja systeemitekniikan laitos

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Ian L. Tuomi		
Työn nimi: Aggregoiva OPC UA-palvelin joustavia tuotantojärjestelmiä varten			
Päiväys:	7. heinäkuuta 2015	**Sivumäärä:**	viii + 53
Pääaine:	Automaation tietotekniikka	**Koodi:**	AS-116
Valvoja:	TkT Ilkka Seilonen		
Ohjaaja:	TkT Ilkka Seilonen		

Joustavista tuotantojärjestelmistä tulee monimutkaisempia samalla kuin niiden tehokkuutta lisätään kehittämällä ominaisuuksia niitä ohjaaviin ohjelmistoihin. Joustavien tuotantojärjestelmien tuottamaa tietoa kerätään enenevissä määrin, ja korkeamman tason järjestelmät käyttävät sitä jatkuvasti uusilla tavoilla. Tämän tuloksena sujuva tiedonkulku eritasoisten tietojärjestelmien välillä on muodostunut tärkeämmäksi kuin aiemmin. Lisäksi joustavan tuotantojärjestelmän laitteiden integrointi on haastaavaa käytössä olevien protokollien määrän vuoksi. Tiedonsiirtoprotokolla nimeltä OPC UA ja aggregoiva palvelinarkkitehtuuri ovat yksi mahdollinen menetelmä joustavien tuotantojärjestelmien viestintästandardien modernisoimiseksi ja integraatio-ongelmien ratkaisemiseksi. Tämän diplomityön tavoite on selvittää joustavaan tuotantojärjestelmään liittyvän aggregoivan OPC UA-palvelimen vaatimukset ja suunnittelu.

Tässä työssä laadittiin joustaviin tuotantojärjestelmiin tarkoitetun aggregoivan OPC UA-palvelimen vaatimusmäärittely, suunnitelma ja toteutus, sekä demonstroitiin sen toimivuus. Vaatimukset määriteltiin perustuen aiempaan kirjallisuuteen. Nämä vaatimukset täyttävä aggregoiva OPC UA-palvelin suunniteltiin ja toteutettiin prototyyppinä. Prototyypin toiminnallisuuden osoittamiseksi sillä kehitettiin tiettyjä käyttötapauksia vastaavia sovelluksia.

Ulkoisen asiakkaan pääsy laitteiden tietoihin, tiedon historiointi, vapaasti määriteltävät hälytykset, ja noodiarvojen yhdistäminen haluttuun toiminnallisuuteen havaittiin oleellisiksi vaatimuksiksi. Aggregoivan palvelimen toteutus OPC UA-standardia hyödyntäen todettiin käytännönläheiseksi. Tässä työssä ei löydetty esteitä OPC UA-standardin käyttökelpoisuudelle joustavissa tuotantojärjestelmissä. Lisäksi OPC UA-standardin hyödyt kasvavat, kun yhteensopivia laitteita, joita ei tarvitse erikseen integroida on saatavilla. Vielä ei kuitenkaan ole selvää, kuinka laajasti OPC UA-standardia kannattaa käyttää eri tietojärjestelmien välisessä kommunikoinnissa.

Asiasanat:	FMS, OPC Unified Architecture, aggregoiva palvelin
Kieli:	Englanti

Preface

I am in great gratitude to my advisor Ilkka Seilonen for his expertise, guidance, and great patience during the writing of this thesis – *thank you.*

Thanks also go to the good people at Fastems for providing context for this work, and for the thought that went into the use cases used for experimentation.

Many people made it possible for me to write this thesis, but the most important ones are my parents. Thank you Mom and Dad, for your love, help, and support throughout my studies and life, and for the sacrifices you have made for me.

I also want to thank all my friends who made the time I spent writing this thesis better. I specifically want to thank Joona for the good company and discussions we had during lunch, Peter and Teemu for being great company at the gym, and Ilmi for being the greatest roommate one could possibly hope for.

Last but not least, a huge thank you goes to all the people of Bratislawa Youghurt for the incalculably positive effect they have made to my time in Otaniemi.

Helsinki, July 7th, 2015

Ian L. Tuomi

Abbreviations and Acronyms

BOM	Bill of Materials
B2MML	Business to Manufacturing Markup Language
CAM	Computer-aided Manufacturing
CAD	Computer-aided Design
CMM	Coordinate Measurement Machine
COM	Component Object Model
CNC	Computer Numerical Control
DCOM	Distributed Component Object Model
DCS	Distributed Control System
DNC	Distributed/Direct Numerical Control
ERP	Enterprise Resource Planning
FMC	Flexible Manufacturing Cell
FMS	Flexible Manufacturing System
HMI	Human Machine Interface
GUID	Global Unique Identifier
IDEF0	ICAM Definition for Function Modeling
JIT	Just in Time
KPI	Key Performance Indicator
LIMS	Laboratory Information Management System
MES	Manufacturing Execution Systems
MESA	Manufacturing Enterprise Solutions Association
MOM	Manufacturing Operations Management
NC	Numerical Control
OEE	Overall Equipment Efficiency
OLE	Object Linking and Embedding
OOP	Object-oriented-programming
OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
PC	Personal Computer

PCS	Process Control System
PDM	Product Definition Management
PLC	Programmable Logic Controller
REST	Representational State Transfer
SCADA	Supervisory Control and Data Acquisition
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TPS	Toyota Production System
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WMS	Warehouse Management System

Contents

Chapter 1

Introduction

1.1 Background

Flexible manufacturing systems (FMS) are a means of production able to produce varying products in small runs while remaining cost-effective. A key difference when comparing FMS to more traditional forms of production, such as transfer lines, is the ability to react to changes. Reacting to expected and unexpected changes requires making real-time decisions concerning manufacture in the FMS itself, and delivering these instructions to each device taking part in the process. Information across devices needs to be combined in order to gain a view of the FMS as a whole, and to orchestrate, schedule and analyze the system efficiently. This requires implementing interfaces to each device, which is considerably less expensive when there are fewer communication standards in use. Additionally, the relevance of the availability and free flow of this information to higher-level systems has increased in the past decades. A single interface to the FMS simplifies this vertical integration for external clients. Reducing the number of used protocols, as well as implementing some centralization on the FMS cell level is therefore desirable.

OPC UA is a protocol for connecting, configuring, and integrating systems. Its specifications have been developed over many years with the intention of being a full-featured solution for a wide variety of industry use cases [12]. OPC UA is gaining popularity as a communication standard in devices that can be used in FMS. This is also indicated by previous work on interface integration challenges in FMS [13]. If this is the case, adopting OPC UA would reduce the number of supported communication standards. The concept of an aggregating server architecture is included in the OPC UA standard [17, sec. 6.3.6].

This thesis is part of the LeanMES project in the MANU research program of FIMECC Ltd. (*Finnish Metals and Engineering Competence Cluster*). The main partner was Fastems Ltd.

1.2 Research Objectives

The overall research objective of this thesis to is define and design an aggregating OPC UA server specifically for use in FMS. The exact requirements of such an aggregating OPC UA server in this application domain are not known, and there are no existing designs for aggregating OPC UA servers for FMS.

Therefore the sub-objectives to this overall research objective are:

- A requirements definition for an aggregating OPC UA server for FMS based on the needs of the application domain learned from a literature review.
- A design complying with the requirements. This includes the design of each required feature, as well as the design for the general software architecture combining the functionality.
- A demonstration of the possible benefits of OPC UA aggregating servers for FMS control applications. To achieve this end, a prototype based on the design is implemented and experimented with by developing a few example applications using it complying with selected use cases.

1.3 Research Methods

The framework which this thesis follows is design science as defined by von Alan et al. [37], a research framework for identifying problems, solving them with artifacts and evaluating the artifacts themselves. According to Peffers et al. [27], the design science process includes six steps: problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation, and communication.

Four artifacts will be created as the result of this work – the requirements, the design, the implementation and the evaluation of the aggregating OPC UA server for FMS. The problem identification and motivation leading to the requirements will be conducted as a literature review of the application domain, FMS, and the used technology, OPC UA. Determination of which parts of the OPC UA specifications are relevant to the research problem, and a description of these parts is in order. Additionally, literature relating to OPC UA aggregating servers is reviewed, in order to learn from approaches to the subject taken in previous work. The results of this review will be the basis for forming the requirements of an aggregating OPC UA server for FMS. Based on the requirements, a design is created to fulfill them. The design is implemented as a proof-of-concept prototype which is subsequently demonstrated. The evaluation consists of a demonstration that the prototype is configurable to adhere to selected use cases, and that it does so by providing functionality stated in the requirements.

1.4 Thesis Outline

Chapter 2 contains a literature review on the application domain concerning this thesis, flexible manufacturing systems. It begins with a brief history, and explains the meaning of flexibility and the devices and software used to achieve it. The relation of FMS to other business processes in a manufacturing company are also explained.

Chapter 3 continues the literature review on the topic of OPCUA. The basic concepts and design of OPC UA will be explained, and the relevant features will be described. The chapter concludes with an overview of previous work concerning the topic of OPC UA aggregating servers.

Chapter 4 will define a requirements definition for the aggregating OPC UA server for FMS in relation to other systems. The chapter starts with an overview of the kinds of functions the system will accomplish, and its role as part of the FMS.

Chapter 5 contains a detailed explanation of the architecture, modules, and functionality that is part of a software design that has been made to satisfy the requirements in Chapter 4.

Chapter 6 describes the tools and environment related to an implementation of a prototype based on the design in Chapter 5, and demonstrates the functionality of it by using it to develop applications according to three use cases.

Chapter 7 concludes this thesis with a summary, conclusions, and suggestions for further work.

Chapter 2

Flexible Manufacturing Systems

The origin of Flexible Manufacturing Systems (FMS) can be traced to the 1980s, when demand for efficient, automated small-scale production was matched with the increasing sophistication of computer systems. Up to that point, only mass production was fully automated, whereas small-scale production relied on expensive machine-assisted human labor. For many end products, it is not desirable to scale production up to drive costs down – the demand for them is too low. Need for a specific part can be intermittent, in some cases even a rare occurrence. The flexible manufacturing system bridges the gap between high- and low-volume production, combining the flexibility of prototyping shops with the efficiency of production lines. The FMS was created to be a mode of production which is inherently flexible, and is designed to achieve cost-effective production under varying demands. This is achieved with a set of computer-controlled, integrated devices capable of storing, transporting and processing the required material. Ultimately, the objective of flexible manufacturing is to increase utilization, throughput and quality, decrease lead times and storage quantities, and improve due date reliability [31].

2.1 Flexibility

There is no universally accepted definition for what differentiates a flexible manufacturing system from a non-flexible one [9]. There are, however several means in which a FMS can be flexible. Browne classifies them as *production*, *product*, *process*, *machine*, *operation*, *expansion*, *routing*, and *volume* flexibility [3]. Flexibility is defined to mean that the system is capable of all or some of these kinds of flexibility. Production flexibility refers to the entirety

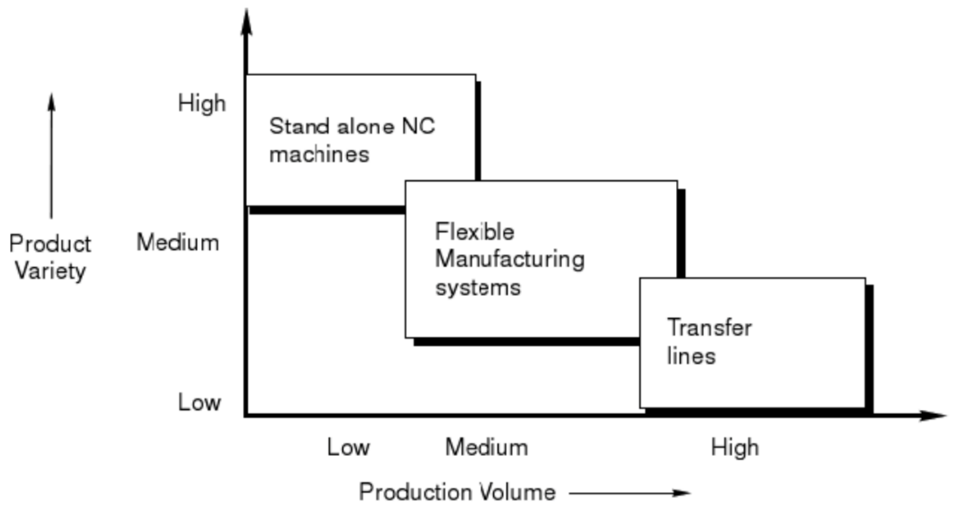


Figure 2.1: In terms of cost-effective variety and volume in manufacturing, FMS occupy the area in between transfer lines and standalone machines[31].

of parts that can be produced – it is dependent on the other types of flexibility. Product flexibility is the ease of configuring the system to economically produce new products using the same tools. Process flexibility refers to the variety of jobs a given FMS can process. Machine flexibility refers to the ease of making physical changes to the system required by production, such as changing tools or replacing worn-out parts. The order of operations in a FMS is not necessarily fixed – operation flexibility is the extent to which the order is changeable. Routing flexibility refers to ability of the processed material to take different routes in the FMS during manufacturing, for example when a certain route has broken down. Volume flexibility is the ability to profitably change production output quantity. Expansion flexibility is the ease building and expanding the FMS by adding components.

Flexibility has costs and can be in conflict with cost-efficiency as a goal [5]. If a high volume of a single product is desired continuously, transfer lines designed for high throughput are typically more economical. Similarly, if product variety is very high, as in prototyping, standalone NC machines may present a more optimal solution. As illustrated in Figure 2.1, FMS occupy an area in between those extremes [31].

Machine utilization is increased by eliminating machine setup, manual intervention and slow transfer times. Part inventory is reduced, and machine throughput increased by intelligent scheduling. Gradual evolution towards increased flexibility and customizability in all kinds of systems is an ongoing trend visible in larger scale manufacturing as well, and many industrial systems not considered FMS can be described as flexible.

2.2 Devices

A FMS can be described as a set of computer controlled machines and storage which are interconnected together with a transport system [31]. Machines able to independently manufacture custom designs, such as CNC machines are utilized as components of FMS, along with automated material handling. This combination significantly reduces the human effort required to manufacture each part and leads to lower reaction times. It further enables a production system to react to changing demands without extensive loss in profit, even when a specific part is rarely needed. The range of components is large, and there are many ways to organize the layout.

Loading Stations

A FMS must have inputs and outputs for materials and finished products. Loading stations are used to load raw or unfinished parts to pallets, and to extract completed products from the system. Stations can also be used for inspecting, washing and assembling parts [31].

Transport

The transport mechanism handles material transport between work stations, storage and machining within the FMS. Cranes are a common transport method for moving materials and manufactured parts between stations and machine tools. Other types of transport devices include robots and conveyor belts [31].

Storage

FMS flexibility is increased by including a storage buffer for raw material and finished products. Otherwise materials need to be loaded and extracted constantly. This is done in the form of an integrated storage warehouse accessible by the transport mechanism of the FMS. Typically storage is done on shelves.

Tool Handling

Automated tool handling adds capability to a machine tool by increasing its machine flexibility. It stores and delivers a variety of tools that may be needed during manufacture. In addition, the tool management system holds details about tool abilities, tracks their state and detects their malfunction. Advanced tool management systems can even replace broken tools automatically. This helps the FMS in achieving process flexibility.

Machine Tools

Machine tools are the devices actually operating the tools to physically alter the raw material. They are typically Computer Numeric Control (CNC) machines. A CNC machine receives precisely encoded program sequences it executes to create desired products with the raw material and tools provided by the FMS. CNC machines can operate by several principles. In CNC mills, the tool rotates and works on the material – in CNC lathes the material, typically cylindrical in form, is rotated.

2.3 FMS Control Systems

A FMS controller observes the devices in the FMS and issues real time commands to them. Communication from the FMS controller to devices consists of numeric control (NC) programs and individual control data downstream, and capacity and quality control data upstream. The FMS controller orchestrates the production process of orders originating from higher-level systems, such as enterprise resource planning (ERP) systems. The FMS controller also holds an inventory of material resources and schedules and tracks device availability. It may record production data enabling traceability or gather process information from the shop floor into reports, concentrating individual sensor-level data into a form that is easier to use in decision-making. The responsibilities of an FMS controller overlap with those of manufacturing execution systems, or MES. MES is a concept of computer software which tracks and controls the transformation of raw materials into finished products. A FMS controller can be interfaced or act in parallel to another control system providing MES functionality, or incorporate part or all of these features and act as the sole control system of the FMS.

As the range of device components increases, so do integration issues. This is an important concern for FMS, where variation in equipment and communication protocols are high and FMS controllers typically need to connect to several types of interfaces. A considerable part of developing FMS controller software consists of the integration challenges due to this. Though internal system integration in a FMS is challenging and expensive due to the amount of equipment involved, it is achievable. The challenges are more prominent when considering integration with external systems. Interfaces in the field are often implemented in non-standard ways, requiring external entities to implement several client-side interfaces in order to connect to each part of the FMS. This drives up costs, takes time, and leads to data that would otherwise be valuable not being exchanged. As configuring external clients to connect to the FMS should simple, it makes sense to provide a standard interface to the FMS, even when a FMS control system is not internally adhering to such a standard.

An industrial communication standard, called OPC UA, detailed in the next subsection, was adopted by this work as a potential solution to these integration issues.

Chapter 3

OPC Unified Architecture

3.1 Overview

OPC[1] Unified Architecture, or as it is commonly known, OPC UA, is a development of older OPC standards, collectively referred to as OPC Classic. Focusing on essential features and restricting APIs to use the Microsoft Component Object Model (COM) and Distributed COM (DCOM) enabled OPC Classic to cut complexity and gain traction quickly. It contains the OPC Data Access[15], OPC Alarms *&* Events[14], and OPC Historical Data Access[16] standards. The limitations of remote access capabilities of COM/DCOM, the platform restriction to Microsoft Windows, as well as a limited feature set prompted the creation of a new standard, OPC UA, which is the topic of this chapter. It was required to incorporate all the previous features of OPC Classic, and ease communication and communication security in distributed systems. OPC Classic was conceived as something akin to a software driver, like a PC printer driver. In the development of the concept of OPC UA, this vision was expanded with increased data modeling capabilities in the form of object-orientation, a type system, complex data, and ability to represent meta-data [12].

OPC UA is grounded in few, but powerful principles. One is the graph data structure, consisting of nodes and vertices. Graphs can be used to represent a wide variety of structured information. Useful parts of object oriented programming, such as type hierarchies and inheritance are also

[1]OPC originally stood for *OLE for Process Control*, where OLE stands for *Object Linking and Embedding*. The acronym has since become convoluted, as OPC UA does not use OLE. In 2011, the OPC Foundation officially renamed OPC to mean *Open Platform Communications*, but this has not caught on. *Open Productivity & Connectivity* is also sometimes used. In practice OPC is simply OPC.

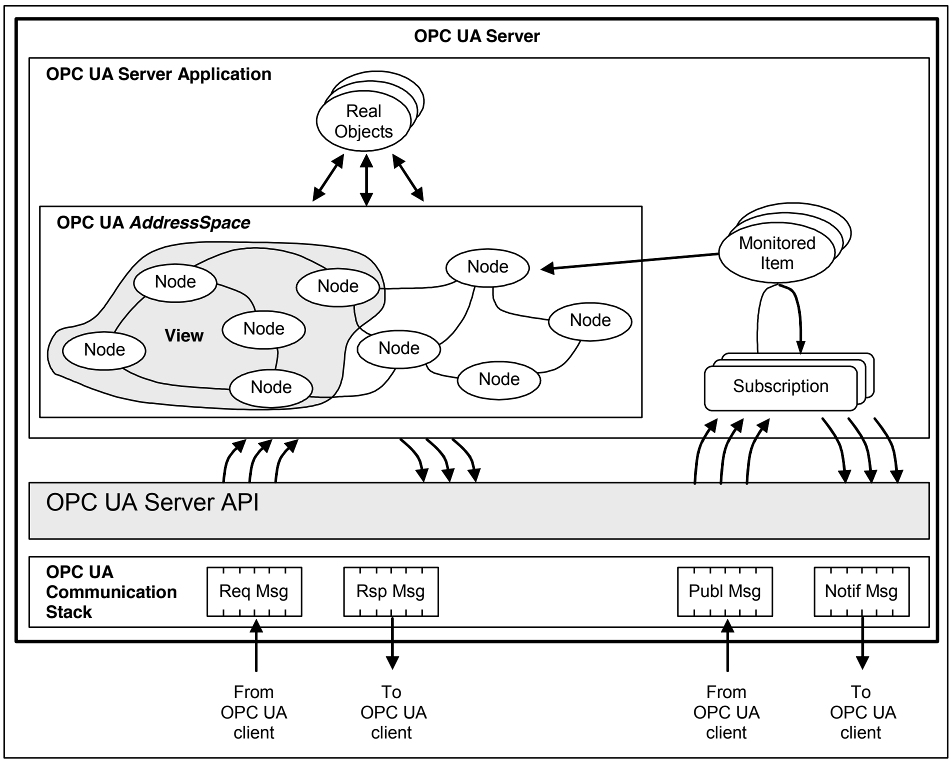


Figure 3.1: Overview of OPC UA concepts [17, p. 14]

.

incorporated into OPC UA. This allows OPC UA clients to handle instances of the same type similarly and avoid specialized data and functions. This is done by relying on rules and conventions the type is governed by. Types can be shared between applications, and can even be used to generate functional source code [10][26]. It is a way of defining the hierarchy, relations, attributes and behavior of the entire automation system. Data transfer using the OPC UA protocol involves at least a client and a server. A server contains the information model, and the client accesses that information model using services [17].

3.2 Used Technologies

The OPC UA standard is defined in an abstract manner, in order for the technology used for implementation to be independent of it. This enables the standard to remain relevant, even if technologies widely in use change. The tasks required to be accomplished by the technologies chosen are data encoding, data security, and data transport. A set of technology that implements this set of tasks is called a *stack*. The OPC foundation provides stacks for several languages. Data Encoding is done in OPC UA Binary or XML. The standard stack implements network communication over TCP. SOAP over HTTP, is also supported, but in practice it is rarely used [22]. Several companies also produce Software Development Kits (SDKs), which include the stack, and more importantly, implementations of the features in OPC UA. This enables developers to develop applications with ease. In practice purchasing a SDK is essential for utilizing the full extent of OPC UA, as implementing these features independently is a major task. Software Development Kits are available in many major programming language platforms, including ANSI C [32], C++ [33], .NET [35], and Java [29] [34]. All support the most important features, but feature coverage is often not full.

3.3 Address Space

The address space of an OPC UA server is the totality of the data it exposes. It is a graph consisting of nodes and edges. Nodes are identified by their *NodeId* and are divided into *NodeClasses*. A node contains its edges, which are called *Reference*s in OPC UA. They connect a node with others. There are six different kinds of NodeClasses: *Object*s, *Variable*s, or *Method*s, which are instances of the types *ObjectType*s, *VariableTypes*, and *MethodTypes*. The exact address space structure is specific to the entity developing the aggregating OPC UA server. Excepting certain nodes that must be present on every OPC UA server, there is no predefined way to arrange the OPC UA address space. This enables each implementor of an OPC UA server to arrange the address space according to their own needs.

NodeIds and Namespaces

Each Node is identified with a unique NodeId. These identifiers can be of three types, Numeric, String or GUID. They are unique numbers, unique strings, and unique 64-bit numbers, respectively. Identifying numbers and strings need to be chosen in a way that does not cause a conflict. A GUID is generated randomly, yet its uniqueness is in practice statistically ensured as it is such a large number. Each NodeId has a *Namespace* string, which further differentiates it from other NodeIds. The same Node Identifier can be used in different NodeIds on the condition they are separated by Namespace. In order to avoid redundant use of memory during runtime, nodes do not contain the Namespace string itself, but rather a *NameSpaceIndex*, referring to the Namespace. The NameSpaceIndex can be converted to the Namespace via a Namespace table.

Attributes

Each Node contains a set of *Attribute*s. The exact attributes a node depend on its NodeClass, but some attributes are common to all nodes. An Attribute consists of its name and value in a certain *DataType*. The DataType signifies how the value should be interpreted. Strings, Integers, and DateTimes are some common DataValues. Binary data can also be saved under the DataType ByteString – this enables arbitrary, and even large data to be contained in the address space [17]. Some Attributes are obligatory for all Nodes: NodeId, NodeClass, BrowseName, and DisplayName. These are used to identify, categorize, and name the Node. All the values related directly to the node itself are encoded as Attributes.

References and ReferenceTypes

An OPC UA address space is effectively a graph, and references are its edges. References form the structure and relations of the address space. As with graph edges, references can be directed or undirected. In the case of directed references it is not required for a server to add inverse references to nodes being referred to, but it is strongly recommended, and a practice typically observed [17]. Each Reference has a *ReferenceType*, represented as a Node in the address space. In practice, the most important distinction between different ReferenceTypes is that some are hierarchical, and some are not. References that have a hierarchical ReferenceType may not contain cycles, whereas references with a non-hierarchical ReferenceType can form any graph. Several networks can be formed in the same address space by using different ReferenceTypes to differentiate between them.

Objects and ObjectTypes

An *Object* is a Node that is used primarily to organize the Address Space in whichever way desired. It does not contain data, other than to describe the Object itself. A common ObjectType is *Folder*, used to group nodes hierarchically. ObjectTypes are used as metadata for external applications, as well as to restrict the configuration of an Object through *ModelingRule*s [17].

Variables

Variable Nodes are used to contain values. Each value has a DataType, depending on the Variable. A simple variable consists of a single node, the *Value* Attribute of which contains the current value of the node. Complex variables can contain child variables, and they can subsequently reference child variables down to an arbitrary complexity [17].

Events

Events are occurrences at a specific point of time, which are received via subscriptions. Events can be, for example, notifications of occurrences in the underlying system, errors, or address space configuration changes [20]. Events are generated by a node specified as an *EventNotifier*, which in turn propagates events according to *HasEventNotifier* references. Events have an *EventType*, which can be used to categorize and filter them. One important use of events is notifications and alarm events generated by *Alarms & Conditions*, which are discussed in more depth below.

Browse

The address space of every OPC UA server is required to contain certain nodes, such as the Server object, and the Objects folder. These, among others, provide an entry point into the address space. In order to discover the address space parented by these entry points, the references they contain need to be followed. Provided a node, Browse will return the references it holds. They can then further be Browsed to reveal the rest of the address space [21]. A noteworthy service related to node discovery is *Query*, which returns nodes across the entire Address Space according to user-defined criteria. However, it is not always supported by OPC UA server implementations, and Browse must sometimes be used instead.

Read and Write

Once a node in the address space has been as been identified by NodeId, its Attributes can be accessed using the *Read* and *Write* services. These services are always used when accessing node Attributes, and their names are self-explanatory. Read and Write will trigger value change notifications for Nodes with Subscriptions[21].

Subscribe

Clients can *Subscribe* to three different types of data changes on an OPC UA server: variables, events, and aggregated values. Each data source is represented by a *MonitoredItem*. Monitored items check values for changes according to a predefined sample interval. They also hold a data or event queue, which has a configurable size. By default, only one data change is queued. For events, the default behavior is to queue as much event data as possible according to server limits [21]. One or more monitored items are combined into a single *subscription* item. The subscription keeps track of the monitored items and delivers information on data changes with the *Publish* service according to a publish interval. A filter can be introduced for each type of monitored item. In the case of variable data, the filter can be a deadband, meaning the rate of change effects whether a notification message is triggered or not. It can also be conditioned on the type of change, for example when the status of the value changes. Event filtering can be even more complex. Aggregate data filtering (*not to be confused with server aggregation*) is a method of sampling time series for averages and the like for set segments of time.

Historical Data

Historical data refers to a time series of changes in the Attribute value of a certain Variable. Variables contain the *Historizing* attribute, which indicates whether historical data is currently being saved. The *HistoryRead* service provides access to the time series. Historical data is typically implemented outside of the SDK. Historical data can also be aggregated, meaning an average or other time series analysis can be performed on it by the server [18].

Alarms & Conditions

The OPC UA specification part 9, Alarms & Conditions, was released in 2012 [23]. It introduces several new concepts that can be used for system monitoring. *Conditions* represent the state of a part of the system and can be triggered to create an alarm. They are instances of *ConditionTypes*, and are necessarily not exposed in the address space. They maintain an internal state, a state machine that can be simple or arbitrarily complex. The complexity of a condition is typically identified by its ConditionType. The Base Condition State Model has two states which refer to its own activity: Enabled and Disabled. When a condition changes to Enabled, that action as well as subsequent actions generate *event notifications*. States can be *Acknowledged*, meaning the operator can indicate that the alarm event has been recognized.

Alarms extend acknowledgeable conditions by adding additional states. An alarm can be *active*, *shelved* and *suppressed*. An active alarm signifies that the condition it represents is happening. A shelved and a suppressed alarm act in the same way, the difference is that an alarm is shelved by the operator, and suppressed by the underlying server. Shelved or suppressed alarms are still fully functional, unlike disabled alarms. Their utility is that they can easily be filtered out by the operator when faced by multiple alarms at once. In this way the operator can concentrate on the important alarms.

3.4 Companion Specifications

The OPC foundation has described *how* data can be described and transmitted from one system to another. Other organizations can further define companion specifications for specific types of information. Several of these specifications, based on the standard OPC UA specifications, have been developed [28] [25]. Models representing hardware should typically be based on OPC UA for Devices [24]. Expansions of existing information models can be general, or they can have a very specific use case. These provide a common substructure for related OPC UA servers and provide guidelines for their development. Provisional information models, based on the standard specifications, can be expanded even further by end users. A shared information model enables vendors and implementors to share application code that is able to utilize the metadata present in information models. Companion specifications are often not utilized in their entirety. Implementors typically use only parts that best apply to their specific case.

3.4.1 OPC UA for Devices

OPC UA for Devices[24], released in 2009, describes a standard way to organize and typify information related to physical equipment, such as sensors, actuators and communication devices. The specification can be used to model the features, settings and the hierarchy between them. The *TopologyElementType* represents a system containing configurable parameters (a *ParameterSet*). A *DeviceType* represents an actual device. It can also consist of other instances of DeviceType, or instances of *BlockType*.

Devices instantiated according to OPC UA for Devices are organized by an

Object called *DeviceSet*. All devices are not necessarily immediate children of DeviceSet. They can be further organized into sub-devices. DeviceSet acts as an entry point for clients connecting to the device address space. The specification includes a device communication model, which adds information on the network the devices are connected to, as well as information on the connections.

3.4.2 OPC UA Information Model for IEC 61131-3

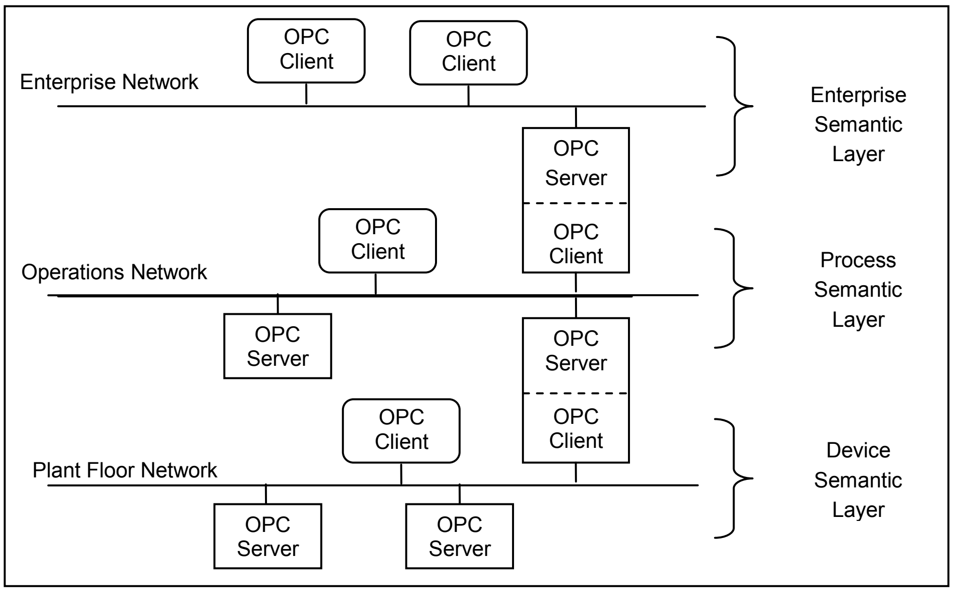
One of the areas where OPC UA is gaining traction is PLC interfaces. The industry standard for defining programs is IEC 61131-3, which defines several programming languages suited for this task, *Structured Text* (ST) and *Sequential Function Charts* (SFC), among others [7]. The PLCopen organization has developed a specification with the OPC Foundation [28]. The objective of the information model is to be able to describe any IEC 61131-3 compliant configuration. However, the constraints of IEC 61131-3 are not taken into account. Therefore, although each valid IEC 61131-3 program can be mapped to a valid OPC UA information model, the opposite is not true [28]. OpenPLC interfaces can be implemented directly into the PLC device [2]. In this case, the OPC UA information model is created according to annotations to the PLC code used to program the device. This is useful in the sense that little additional coding needs to be done in order to create the OPC UA server. This requires the OPC UA address space to adhere to the structure of the code.

3.4.3 OPC UA for ISA-95 Common Object Model

The ISA-95 standard provides a common and flexible vocabulary for modeling information related to enterprise and control systems in industry [8]. It has been specified primarily in order to ease integration between the MES and ERP levels. The goals include reducing incompatibilities and cost and subsequent risk in implementing interfaces. In practice, it aims to bridge manufacturing with business. A partial functional ISA-95 specification in OPC UA has been made by the OPC foundation [25]. The parts of ISA-95 that were deemed most important by the work group and subsequently implemented are Personnel Information, Role Based equipment information, Physical Asset information, and Material Information.

The material model is deemed important for this work. It is used to classify what kinds of materials are handled, and to describe what materials

Figure 3.2: Vertical integration using OPC UA[17, p. 16].



are actually being handled. Material classes classify materials on the highest level, and material definitions can be used to sub-classify them further. Each material class and material definition can be amended with properties describing properties of the classification. The inventory of raw, finished and intermediate materials are represented as lots, which can be further divided into sublots if needed. Material lots often have properties associated with them, such as their location, or results of quality control tests [25].

3.5 Aggregating Server Architecture

3.5.1 Overview

The OPC UA specification includes the concept of a layered architecture of servers, illustrated in Figure 3.5.1. The network is divided into semantic layers, for example enterprise, process, and device layers. A combined OPC UA server and client provides access from higher layers to lower layers – data from lower-layer servers is gathered and aggregated into higher-layer data constructs. The information is concentrated into an interface providing a single access point for higher-level clients [17, sect. 6.3.6]. This architecture is the basis for what has later been named aggregating server architecture. A client accessing the aggregated machines only needs to connect to one server in order to gain access to the functionality it requires. When no values are derived, and the address space is simply passed through as is, the server is referred to as a *chaining* server [17]. Aggregated servers can further be aggregated, for example in a production facility containing several FMS [11].

The interface clients connect to in order to access the functionality of the aggregating server is the aggregating address space. Nodes in the address space of the aggregating server can directly refer to nodes on underlying servers. They can also contain a value derived from several sources. The aggregated address space can also contain alarms monitoring part of the aggregated devices address space, and depend on values from several different devices. The nodes in the aggregating address space can be connected to underlying devices with a mapping. The mapping is configured according to a configuration which specifies the aggregated servers and how their address spaces are mapped into the aggregating address space [4].

Aggregating or chaining servers often include an implementation of OPC UA historical data access. Such a server is often called a historian. A historian collects and logs a specific part of the information generated by servers, and enables it to be retrieved later. The historization of data on a chaining OPC UA server has been the topic of work by Asikainen [1]. The design was found to be applicable to historizing data on an aggregating server. With historization, the aggregating server can deliver persistent data, as well as the results of analysis on that data, to higher-level systems. Aggregate data based on current and historical values is processed on the aggregating server to the extent possible. The amount of network traffic transmitted can be significantly reduced when the transmitted data has already been processed, especially when processing large time series data [6]. Each external system can access the same information directly, not having to calculate the data independently. Centralizing data processing also ensures that key values consumed by each client are calculated in the same way.

3.5.2 Information Models for Aggregating Servers

It is not compulsory to extend the OPC UA information model in order to implement server aggregation. However, encoding configuration information directly in the address space is compelling, especially if an aggregation information model can be standardized. This has been attempted by Großmann *et al.*, who define two information models, one for aggregation servers, and one for aggregated servers [4]. The aggregation server information model contains the servers available for aggregation, which can be populated either manually or by the Discovery service in runtime. Each aggregable server Object contains a Method, which launches the aggregation of the server. The aggregated server information model in turn contains rules to aggregate the Types and Instances it contains. The way the rules are encoded and used is an interesting research problem, but Großmann *et al.* do not specify their solution in much detail [4].

3.5.3 Existing Aggregating Servers

There are a few commercial OPC UA servers on the market which can be said to be aggregating. However, none of them implement an aggregating server in the meaning of the word used in the title of this thesis.

- The Prosys OPC UA Historian [30] connects to several servers and stores values and event data from them using OPC UA HDA. It collects the values and event data from all the servers it connects to, and stores it into a SQL database for later use. It advertises support for unlimited data sources.

- Unified Automation UaGateway is a wrapper *&* proxy which can be used to migrate existing COM/DCOM-based OPC applications to OPC UA [36]. It is aggregating in the sense that it can connect to several servers and provide a single point of access, but its main use case is migration from classic OPC to using OPC UA, rather than functioning as an aggregating server in the sense pictured above.

Chapter 4

Requirements

In this chapter, requirements for the aggregating OPC UA server for FMS are identified, and the systems it interacts with are defined. Requirements are based on work on OPC UA aggregating servers in general, and work on aggregating servers in FMS. The specific requirements of FMS will also be considered in themselves. Section 4.1 defines the systems the aggregating server interacts with, and the role of the aggregating server with respect to external systems. Section 4.2 defines what functionality is required from the server during its lifecycle, and Section 4.3 defines what information the aggregating server is required to contain and process.

4.1 System Definition

The basic system architecture is based on aggregating architecture as described in the OPC UA specifications [17] and by Mahnke [12]. An OPC UA aggregating server connects to devices on the factory floor and allows external clients to connect to it. This is illustrated in Figure 4.1. The aggregating server connects to devices in the local FMS network, which are assumed to contain an OPC UA server. If a device does not contain an OPC UA server, it can be amended with an adapting gateway server implementing the device interface and providing access to it by OPC UA clients. Any OPC UA client can connect to the aggregating server, provided it is authorized. These may include an operator of the system, a system provider, an external service company, or some internal service. Alternate interfaces can also be provided to further improve accessibility to the server, for example using a REST interface.

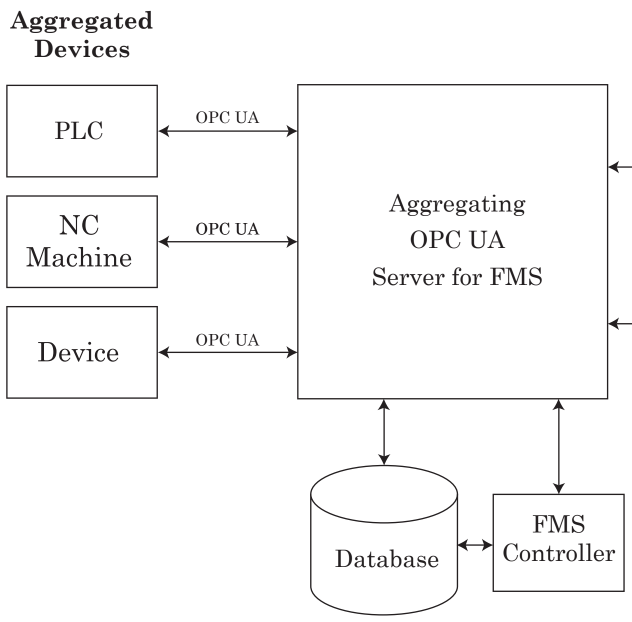


Figure 4.1: The OPC UA aggregating server for FMS connects to FMS devices and allows external clients to connect to it.

The role of the aggregating OPC UA server as defined here is to be supplementary to an existing FMS controller. It primary function is to observe the process in the FMS. It hardly is reasonable for a FMS provider developing an existing FMS controller to change all of its interfaces to OPC UA immediately, when there is no immediate value [13, sect. 4.1.1]. Therefore OPC UA is likely to be adopted incrementally, in new and parallel interfaces to existing ones. The aggregating OPC UA server interfaces with an existing FMS controller via a relational database that the FMS controller utilizes.

4.2 Functional Requirements

Configuring the aggregating OPC UA server requires identifying devices in the FMS in order to connect to their address spaces via OPC UA. In addition, relevant nodes in devices' address spaces must be identified for access with OPC UA services. The configuration-time requirements for identifying nodes do not differ between devices due to each OPC UA server functioning in a standardized manner, according to OPC UA. However, the OPC UA address spaces structures of devices vary – the aggregating server needs to identify relevant nodes, regardless of address space structure. The information required to do this must be included in a configuration that can be modified to suit each FMS and use case involving them. Custom functionality for observing and transforming specific parts of the aggregating server's address space also needs to be configurable.

The NodeIds of nodes are unique to the address space containing them. When representing nodes from several OPC UA address spaces in one aggregating address space, the possibility of a NodeId conflict arises. The aggregating OPC UA server needs to be able to distinguish nodes with the same NodeId, but different address spaces.

In addition to OPC UA devices, the aggregating OPC UA server connects to typically one database, containing information related to the functioning of the FMS, for example material information. Information in different FMS databases is arranged differently. Therefore the configuration describing the mapping between the aggregating server's address space and database schema must be custom-made. Typically, tables are organized into a hierarchy, with rows connected by foreign key values. Some amount of functionality assisting configuration in basic cases should be provided, though it cannot be guaranteed to apply to every FMS database schema describing information.

Functionality during operational time is provided with OPC UA communication, using services. Clients read values with the *Read* service, which the aggregating OPC UA server interprets according to which node the client accesses in its address space. Reading the value of a node can, depending on context, trigger one or more further OPC UA calls to devices. The result can be passed through or processed by the server, according to the context of the call. Using the OPC UA service *Write* similarly initiates a corresponding Write call to a device or database, if applicable. Subscriptions can be made to any part of the OPC UA aggregating server's address space, and the aggregating server handles value changes on aggregated devices as expected, with OPC UA event notifications to subscribed clients. In addition to node value changes, alarm events can also be subscribed to.

FMS providers or support personnel need to occasionally remotely troubleshoot problems at the customer site, or perform other maintenance. The security features of OPC UA enable an OPC UA server to be practically accessed directly outside a local network, even through firewalls [19]. This feature of OPC UA is a significant benefit compared to other communication protocols – it provides value when a FMS provider needs to connect to a FMS it has installed on a customers premises [13]. The aggregating server implements these security features.

4.3 Data Requirements

In addition to information sourced from a database, essentially all information generated by the aggregating OPC UA server is dependent on data contained by devices. The aggregated devices each contain an OPC UA server containing an address space. The address space structure of devices is in practice static – new nodes or references are not created during runtime. Therefore the aggregating server needs to browse through the structure only once. Changing values are, with the exception of timestamps and other time-dependent metadata, contained by the value attributes of Variable nodes. Variable nodes describe the data type of the value it contains. Variables that are of interest are typically integers, booleans, or floating point numbers. The rate of change for Variable values depends on its function in the device. Values such as sensor readings may change very frequently, others may not change at all.

A relational database is another source of information for the aggregating server. The data contained in a relational database is arranged into tables according to a schema. As the aggregating server is being developed for an existing database attached to a FMS controller, the schema is not known beforehand from the perspective of the aggregating OPC UA server. Therefore the aggregating server does not assume any schema, and can be fitted to a wide variety of relational databases.

Observing the FMS often requires more information than the current state. Time series information is needed, sometimes spanning a considerable period of time. For this reason the aggregating OPC UA server is able to represent, retrieve and persist time series data. In order to save and persist values as per OPC UA Historical Data Access, nodes are assigned identifiers sufficient to consistently identify them between sessions. Their time-series data stored in a database is accessed according to these identifiers.

The OPC UA aggregating server acts as an interface to external clients accessing the FMS. The aggregating OPC UA server should allow access to the address spaces of aggregated devices directly, passing their address space through as is. This enables clients to the aggregating OPC UA server to perform all operations in the FMS that would otherwise require connecting to devices directly. It is cost-effective, as it comes at practically no effort – if only passing values through meets the requirements of the stakeholders, no further configuration is required. It is also practical, as the totality of servers and nodes across a FMS can be assumed to be small.

Clients connecting to the aggregating OPC UA server often need to combine several values, sometimes across devices. This functionality is best provided by the OPC UA aggregating server for the FMS. In this way, the calculation of key values has a centralized implementation, saving each client from implementing the same functionality. The aggregating server is able to derive combined values based on multiple values in the aggregated address spaces.

Operators of the FMS diagnose future and present system failures by intuition gained through prior experience. Translating these intuitions into concrete rules that can be executed in an aggregating server frees the operators from redundant work in diagnosing these problems. Diagnostics involving multiple devices can enable the system to keep working even when a single measurement in itself implicates a possible malfunction. Predicting problems based on anomalous events or values can enable an operator to be alerted before a potential equipment failure. This may decrease system downtime resulting from unscheduled maintenance breaks. When a potential problem which cannot be taken into account automatically is detected, it is important to notify users. This is best facilitated in an aggregating OPC UA server by using OPC UA Alarms. Since alarm logic can be complex, it is recommended that defining alarm conditions is as flexible as possible. Alarm severity, description, and other details are configurable in order for alarm event information to be as useful as possible, and to make event filtering useful. The alarms are to be conveniently grouped in the address space so that only alarms relevant to each client can be conveniently subscribed to. Alarm conditions depend on values on the aggregated servers or the connected database. If the values are polled constantly, the system load is increased. Therefore alarm evaluation is event-based when possible.

Most companies using a FMS also have higher-level systems consuming information generated by the FMS. Having direct access to FMS-level data via an aggregating server could aid the business to discover opportunities to increase productivity and to see potential points of failure. Subsequently, the FMS itself can be operated more efficiently. Therefore integration with higher-level systems is a priority for the OPC UA aggregating server for FMS. Melander suggests that standardized OPC UA companion specifications be used to organize data in the FMS OPC UA address space to the extent possible in order to facilitate and ease integration [13, Sect. 4.2.1]. Therefore the aggregating server incorporates OPC UAcompanion specifications in the aggregating server when deemed useful.

In the future, an aggregating server may also provide a platform for gathering usage information. When a FMS is supplied by a company concentrating its efforts on making FMS, it makes sense to gather information on how users are actually using the system. The supplier can analyze the information in order to understand how the customers are using the product. This enables the supplier to know which aspects of product development are actually important. Understanding how customers misunderstand intended ways of using the product can also be beneficial. This knowledge can be used to improve product usability to help proper usage of the FMS. It can also prompt improvements to documentation and training. As there are likely to be many clients, there will be a significant amount information generated in this way. All improvements made based on this information will directly benefit all users.

Chapter 5

Design

This chapter proposes a design of an aggregating server complying with the requirements presented in Chapter 4.

5.1 System Architecture

The aggregating OPC UA server for FMS is an application run as part of the FMS as illustrated in Figure 5.1. Connected devices in the FMS acting as information sources are called *aggregated*, in contrast to the aggregating server. The aggregating server contains OPC UA clients able to connect to the OPC UA server of each aggregated device and access their address space. It also contains one OPC UA server that external OPC UA clients can connect to in order to access the address space of the aggregating server itself. In between the clients and server there is a mechanism connecting them, called a *mapping*. The mapping describes how the data underlying each node can be retrieved, connecting nodes in the aggregating address space to a variety of sources, including nodes in the address spaces of aggregated servers, database queries, and functions combining values from different parts of the aggregating address space. The aggregated address space and the mapping are generated according to a configuration. The configuration specifies what servers are to be aggregated and identifies relevant nodes on the aggregated servers. It further describes the transformations and combinations performed on them and the nodes that are generated in the aggregating address space to provide an interface to this functionality. The aggregating OPC UA server also interfaces with typically one database used by the FMS controller to map a part of its contents into the aggregating address space. A database used to store and retrieve historical data is also interfaced with.

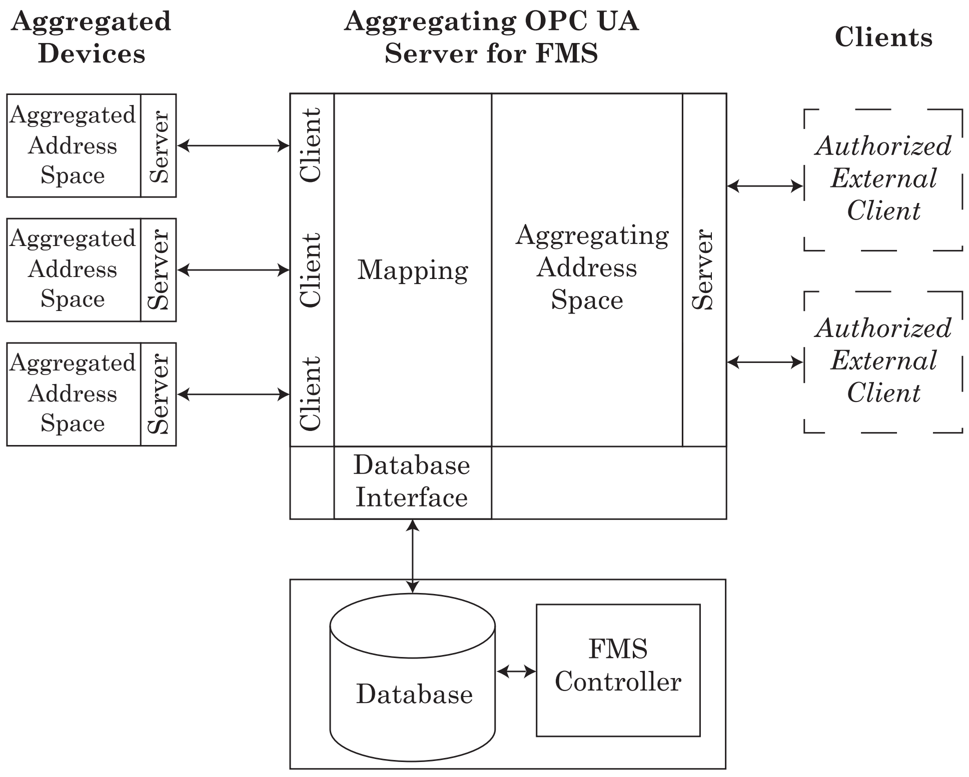


Figure 5.1: The architecture of the system design. All clients and servers are OPC UA clients and servers.

The functionality required from the database – procedures and schema – are part of this design. In Figure 5.1, the same database fills both use cases.

5.2 Modules

5.2.1 Server and Clients

The aggregating OPC UA server contains several clients connecting to aggregated devices, and one server providing access to the aggregating address space. Both clients and server are standard OPC UA software components. One client is created for each aggregated device and is used to scan the address space on the aggregated device. The client is also used to access the device during operation.

5.2.2 Address Space

The address space of the aggregating OPC UA server is the interface external clients connect to. Its structure and contents are illustrated in Figure 5.2. For lack of a need for a more complicated structure, a simple structure was created. As applications of aggregating servers become more specific, more complex address space structures can be created at will. However, complexity in the address space is not desirable, and there has to be benefits to creating a deeply nested address space structure. Certain parts of the address space are static, such as folders directly under the Objects folder, used for organizing the aggregating address space. Others are specific to each instance of the aggregating server and are generated based on the configuration and the contents of aggregated devices.

DeviceSet contains object nodes representing each aggregated device, as per OPC UA for Devices. Each of these nodes organizes a duplicate of the address space of each aggregated device, starting from the Objects folder. This is achieved with minimal configuration. Information is not processed – OPC UA calls are simply passed through the aggregating server and back. The exception to this is the namespace indexes. Each aggregated server contains a namespace table, which follows its own namespace indexing. Therefore in order to avoid conflicts in the aggregating namespace, namespace index attributes read from devices' address spaces are changed to comply with the aggregating server's namespace table. The duplicated address spaces on the aggregating server provide direct access to the address spaces of underlying devices. Explicitly configured nodes outside DeviceSet amend this baseline configuration.

The **Database** folder contains information sourced from a database used by the FMS controller. It can contain individual nodes accessing single values resulting from queries or procedures, or hierarchies of nodes corresponding to structured data in the database. Material information contained in the database can form such a hierarchy. In case such a hierarchy is represented on the aggregating OPC UA server, it is contained in the Materials subfolder of the Database folder. The node types and hierarchy in it correspond with the OPC UA for ISA-95 standard object model [25]. Materials are divided into MaterialClasses, which are represented as object nodes of the ISA-95 type MaterialClass directly under the Materials folder. These nodes contain information on what kinds of materials can be found in the address space, and they can further contain references organizing MaterialDefinitions, which are sub-categories of MaterialClasses. Both MaterialClasses and MaterialDefinitions contain ISA-95 properties represented as variable nodes they reference. Actual instances of materials are represented as nodes of the type MaterialLot, and they are organized under the MaterialDefinition node defining them. MaterialLots also contain the properties describing them, represented as variable nodes they reference.

The **Alarms** folder contains the alarms as standard OPC UA alarm nodes as per OPC UA alarms and conditions. Alarms can be subscribed to individually, or the entire organizing alarms folder can be subscribed to by an OPC UA client wishing to be notified of all alarms. The alarms may also be divided into subfolders allowing users to subscribe to events according to that arrangement. The **Aggregates** folder contains nodes mapped to functions providing combined values.

5.2.3 Mapping Application

The mapping defines the connection between a node in the aggregating address space and its sources. It is created according to a configuration and used during runtime to retrieve and compute requested values. The mapping itself is a data structure enabling efficient addition and lookup of the source of a given node. Since services are used frequently, the performance of searching for a NodeId in a mapping is an important consideration. Hash maps were selected as the data structure for their constant-time performance on lookup and insertion operations. Each hash map maps a NodeId in the aggregating server's local address space into a data source. A node can be mapped to many kinds of sources, part of which are illustrated in Figure

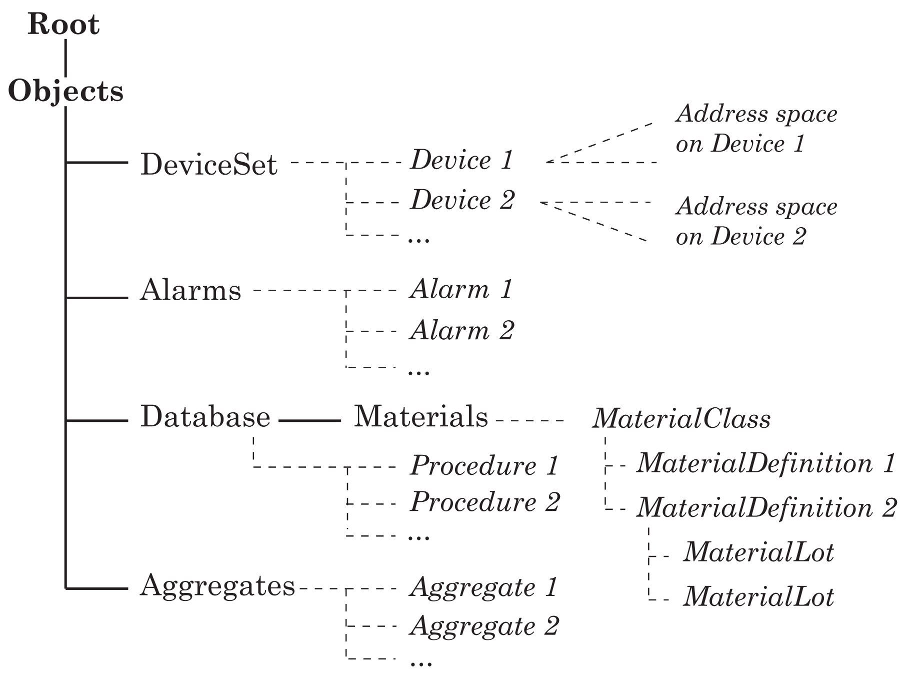


Figure 5.2: The address space of the aggregating server. Bold nodes are present on every OPC UA server. Normal-weight nodes are present in every aggregating OPC UA server for FMS adhering to this design. Italic nodes are specific to each instance of the aggregating server.

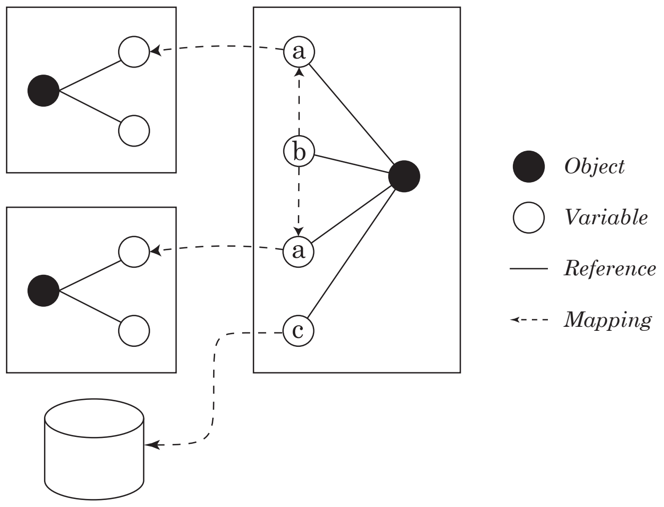


Figure 5.3: A simplified illustration of possible variable mappings. Variables marked with *a* are mapped to external variables. The variable *b* is mapped to a combined value of the variables marked with *a*. The variable marked with *c* is mapped to a database query.

5.3. The source can be a single data point in a system external to the aggregating OPC UA server, such as a node on an aggregated device, or the result of a database query. Nodes mapped in this way could be referred to as proxy nodes, as their aim is to simply represent the external value in the aggregating address space. In this design, all external data sources are first mapped as proxy nodes, and only then further processed inside the aggregating server. When combining values, the mapping contains the method used to combine them as well as their sources, in a function, created during configuration. This function, returning an OPC UA datavalue, queries different parts of the address space, computes the result and returns it. Functions aggregating data can also return a boolean value, and be evaluated during runtime as conditions for activating an alarm.

5.2.4 Database Interface

The database interface enables the mapping to connect nodes in the aggregating server's address space to a database. Mappings can be made to database queries, which can refer to single cell values in certain tables, or be the result of a more complex operation. The database interface is also

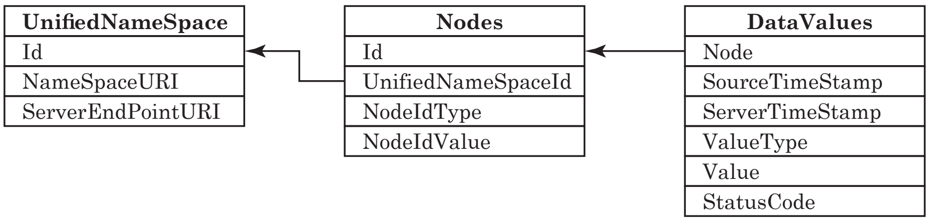


Figure 5.4: Database schema for historical data access.

utilized to map material information contained by the database into the aggregating server's address space according to ISA-95. It is also used to implement OPC UA HDA.

The requirements specify the aggregating server is able to consistently identify nodes, save, persist and retrieve their values, and to present them in the aggregating address space according to OPC UA Historical Data Access [18]. The design follows previous work done by Asikainen [1], with some minor changes. The schema is amended with information on NodeId types and DataTypes to aid node identification and deserialization of stored values into their proper OPC UA DataTypes on the aggregating server. Tables implementing HDA, illustrated in figure 5.4, contain the information required to identify nodes on OPC UA servers, as well as to store and retrieve values.

Nodes are identified in the database by their NodeId and the ServerURI of the server containing them – the same values used to identify nodes in configuration. The NodeId consists of an identifier, an identifier type, and a namespace. Each time a node is historized, the serverURI of the server containing it, and its namespace are searched for in the UnifiedNameSpace table. Each encountered unique combination of namespace URI and server URI are assigned a unique identifying value in the UnifiedNameSpace table (Figure 5.4). This unified namespace identifier defines a certain namespace on a certain server. It is used to unequivocally connect a NodeId identifier and type to a unique node identifier in the Nodes table (Figure 5.4). The NodeId type has to be included because the interpretation of a serialized NodeId identifier depends on it. The NodeId corresponds to the NodeId where the canonical representation of that information resides, whether on an external server or on the local machine – not to a proxy node. Once the unique node identifier has been found, it can be used to insert and select historical values from the DataValues table (Figure 5.4). The DataValues table holds the actual time series data – values, timestamps, and status codes of the Variables which are being historized. The DataType of the Variable is also included, in order to deserialize the saved value on the aggregating server.

The database internal node identifier for each historized NodeId is found during the initialization of the aggregating server – this identifier is used to map HDA functionality to the database. Insertion of historical data to the DataValues table happens on value change indicated by a subscription made on the historized node. The data change event handler of the subscription is set to be a function containing a query inserting the changed data to the DataValues table. Aggregating server historical data access is overloaded to retrieve information from the DataValues table, also using a query that is constructed based on the node identifier used internally by the database. The resulting database operations are direct, and don't require reidentifying nodes during runtime. Users can access HDA on the aggregating server according to standards, and the behaviour of the feature is as expected.

5.3 Functions

5.3.1 Configuration Time Functionality

The aggregating server contains a configuration, describing the OPC UA servers the aggregating server expects to connect to. Servers are identified by their ServerURI, which identifies them globally [22, sect. 5.4.2.1]. Thus the information required to establish connections is a list of ServerURIs of the servers that are to be aggregated. On reading the configuration, a client is created for each server and a connection is established. The clients are used to scan the address spaces of each aggregated server, starting from their Objects folder, and their address spaces are duplicated into the address space structure detailed in Section 5.2.2. As the nodes are created in the aggregating address space, each are added to a mapping specifying which ServerURI and remote NodeId the proxy node is connected to. This enables service calls to those nodes to be routed to underlying devices later on. Additionally, the configuration specifies database procedures used for mappings. Upon reading the configuration, proxy nodes are created in the Database folder and a mapping is created to a function connecting to a database, ultimately retrieving the value.

Once proxy nodes have been created, configuration moves on to create mappings for aggregate nodes and alarms. In order to create these mappings, individual nodes are identified by their NodeId in conjunction with the ServerURI of the server they are mapped to. A NodeId is enough to unambiguously recognize a Node as each NodeId on a given server is unique [20, sect. 5.2.2]. The configuration for an aggregate mapping identifies the nodes to be mapped to, and a function which takes their values as input and outputs a single value. In the case of node mappings, that value is a DataValue. For mappings to alarm conditions, that function returns a boolean value.

When a node is configured to be historized, its NodeId, address space and originating server are sent to a database procedure, which determines the identifier for the node used internally by the database. The identifier is created on the database server if the node has not been historized before. After the historized node is identified in the database, the aggregating server makes a subscription to it with an internal OPC UA client. This subscription is used during runtime to identify value changes, which in turn trigger saving the value into the database. A mapping specific to retrieving historized values is made, which can be called on to retrieve time series data from the database.

The requirements specify using the OPC UA implementation of ISA-95 to present database material data in the address space of the aggregated server. Each FMS controller saves material data in its own way, and assigning ISA-95 types to an unknown database structure automatically is a hard problem. Therefore the mapping to ISA-95 must be custom-made each time. However, the ISA-95 specification contains hierarchies of types, and relational databases often contain comparable hierarchies as well. When a relational database contains a structure where tables form a hierarchy where rows are connected by foreign keys, a relation can be found to the ISA-95 hierarchy of types. In this case, each row of a table is mapped into an OPC UA object node, which is assigned the desired ISA-95 type. The nodes are named according to a column chosen to be most descriptive. Other attributes on the node's row are mapped to the address space as variable nodes and assigned the proper ISA-95 property type. The function these variable nodes are mapped to is a database query fetching the cell value from the table according to the corresponding row id and column name The column name related to the cell specifies the variable's name, and the value of the cell is mapped to the variable's value. If the mapped rows contain a foreign key specifying a relevant hierarchy, the referred table can also be mapped in the previous manner, using the ISA-95 subtype of the type previously used.

The requirements specify the aggregating server implements configurable alarms. Users configure alarms by writing their logic directly in application code. The condition triggering the alarm is a function returning a boolean value. It typically reads node values and calculates the outcome based on them. The responsibility of the alarm manager is to evaluate alarm conditions and trigger alarms as needed. Since alarm conditions may be expensive to evaluate, nodes the alarm condition depends on are also passed to the alarm manager. These nodes are subscribed to, in order for alarm conditions to be evaluated only when values they depend on change. The source for the issued alarm events are the alarm objects themselves, which are contained in the Alarms folder. The alarm objects contain references to the Alarms folder containing them, that according to OPC UA, propagate the events to the Alarms folder. Thus subscriptors to either alarms themselves or the alarms folder are notified by alarm events.

5.3.2 Runtime Functionality

When the aggregating server is running, it responds to OPC UA communication to its address space by executing mappings. Additionally, it monitors event notifications, saves values for historized nodes, and evaluates conditions for alarms. Each time the Read service is invoked on a node on the aggregated server, the remote mapping is checked to see if there is a mapping for this local node. If so, the mapping yields a client connected to the mapped server and the remote NodeId of the source node. The read service is run on the client connected to the remote server hosting the address space, containing the namespace with the source node. The NodeId of the source node, contained in the mapping, is passed as a parameter. The returned result is passed on to the client invoking the read service on the proxy node. Runtime performance can be increased by mapping only the Value attribute, and caching all other values, not likely to change, directly into the aggregating address space.

Nodes may also be mapped to functions returning a valid value, in other words an OPC UA DataValue. If so, an arbitrary function, provided during configuration is executed and its result returned. The function can combine values from other nodes in the address space or fetch values from custom interfaces. The sequence diagram of a Read call to a Node on the aggregating server which returns a combination of values on separate devices is illustrated in Figure 5.5.

Mapping to database queries is a special case of mapping to functions.

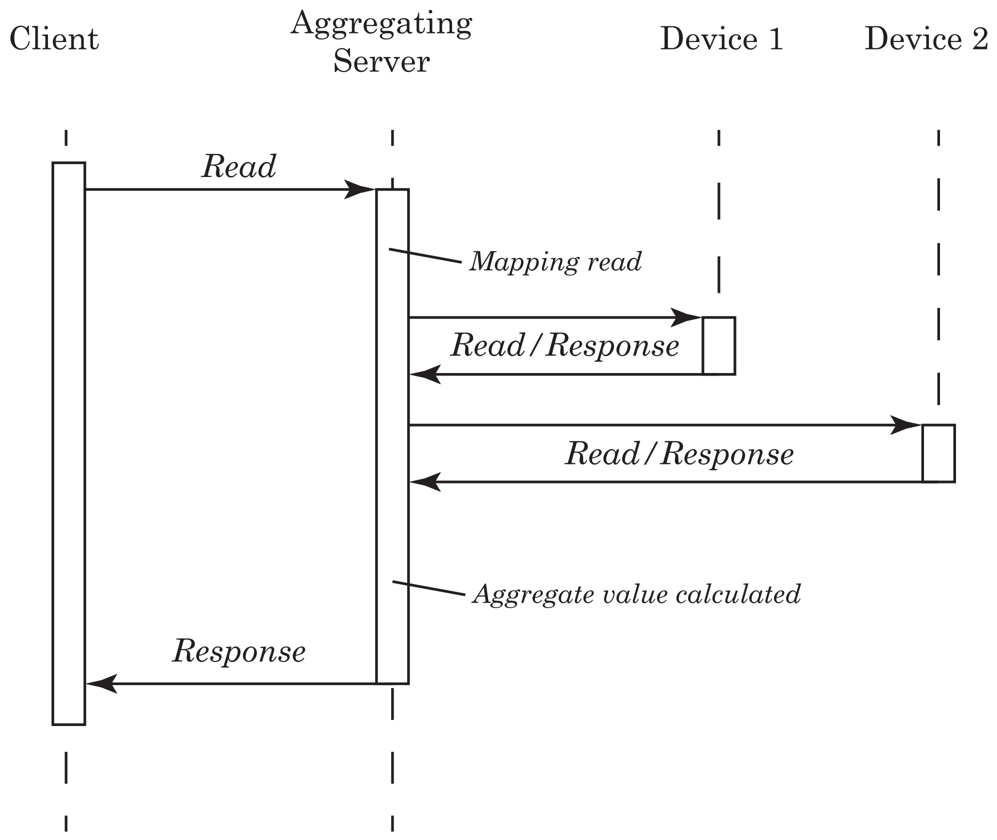


Figure 5.5: A Read call to a node combining values on several devices.

The mapped function initiates a database query, and returns the value from the server. The query can be constructed to refer to a certain unique value in a certain table, or it can be the result of a stored procedure on the SQL server. If a stored procedure is referenced, that procedure has to be created on the SQL server beforehand.

During runtime, changes to historized nodes are detected via data change notifications sent by a subscription made during configuration. When a historized value changes, that value is immediately added to a database containing the historizing values. Similarly, when the history of a historized node is read, the database is queried. Reading the history is mapped to a database query selecting the value series from the database.

Alarm conditions, provided during configuration, are tracked and evaluated by an alarm manager. Alarm conditions depend on one or more values on aggregated devices. The alarm manager subscribes to each value alarms depend on and evaluates an alarm condition when one of the values the alarm depends on has changed. When alarm conditions are met, the alarm manager issues an OPC UA alarm event which subscribed clients receive.

Chapter 6

Implementation and Experimentation

The implementation of an experimental prototype complying with the design presented in Chapter 5 is presented in this chapter.

6.1 Application Development Platform

6.1.1 Overview

The development environment, running in its entirety on a PC, is illustrated in Figure 6.1. Aggregated devices are represented by OPC UA servers run in the Beckhoff TwinCat PLC development application [2]. TwinCat is capable of running a local OPC UA server which is functionally identical to OPC UA servers on Beckhoff PLC devices. PLC code containing data structure and variable definitions was loaded into TwinCat. Based on this code, an OPC UA address space was generated automatically. A database, Microsoft SQL Server 2008 R2, providing historical data access functionality, was also installed and run on the PC. Procedures and tables were added to the database to implement the OPC UA HDA functionality. The same database was used to represent a FMS controller database with material information. Tables duplicating the schema of a real-world FMS controller were created in the database and populated with test data. UAExpert was used as a client interfacing with the aggregating OPC UA server. UAExpert, developed by Unified Automation, is capable of discovering, connecting to, and interfacing with OPC UA servers.

The language C♯ was used for creating the prototype aggregating OPC UA server itself. This lead to the use of the Unified Automation SDK [35]. Standard components of the SDK, such as OPC UA clients and servers, were extended to provide the mapping functionality. Most functionality is im-

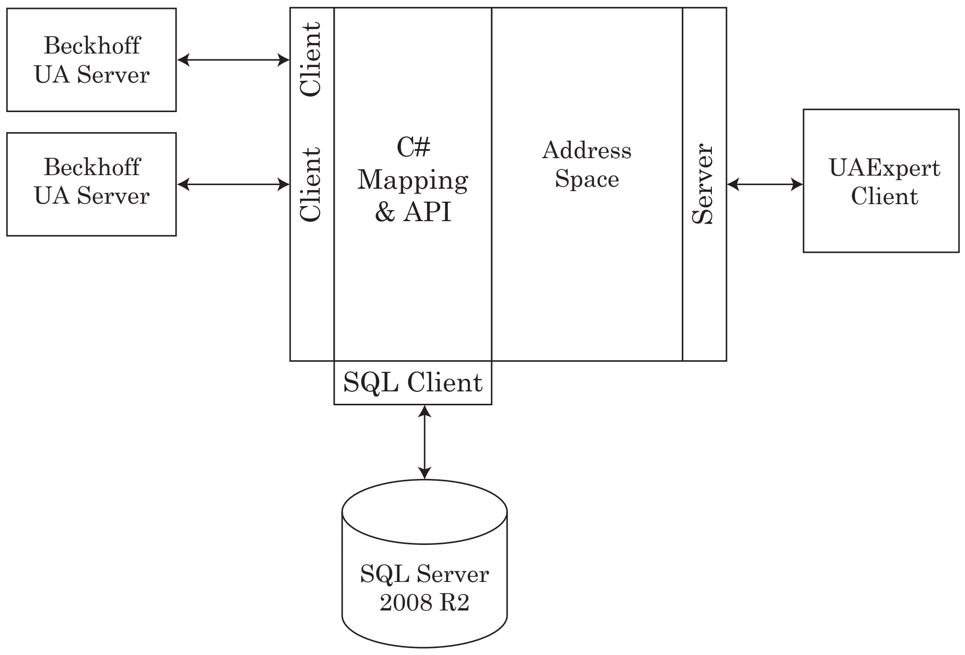


Figure 6.1: The development environment of the prototype. All components are run on a single PC used for development.

plemented by overriding OPC UA service implementations in the SDK class NodeManager to alter the default behaviour. The overridden functions are supported by utility classes providing database connectivity, data structures for mappings, and run-time monitoring.

6.1.2 Application Development API

The configuration was mainly written in code utilizing an API, and compiled as part of the binary executable. However, configuration of aggregated device URI in the prototype is done interactively during startup. Since the aggregated OPC UA servers were running locally, the full URI is not required. Only port numbers are entered into the text-based interface. For any implementation involving external OPC UA devices, configuration might take place with the Discovery service, or by listing the URI in their entirety in the configuration.

Other configuration of the aggregating server is done by calling configuration functions. Remote nodes are mapped by default, so they are not mapped by the user.

- The function `mapNodeValue` is called to map nodes to custom functions. The parameters are the NodeId of the node, and a function returning a DataValue.
- The function `mapHDA` selects a node to be historized according to HDA. Its only parameter is the NodeId of the node to be historized.
- The function `addSQLNode` creates a node in the database folder and maps it immediately to a specific procedure in a specific database. The name of the node, the procedure name and the database connection string are provided as parameters to the function.
- The function `createAlarm` creates an OPC UA Alarm. It takes as parameters its name, source node, and an alarm condition function returning a boolean value indicating whether the alarm condition is fulfilled.

Material information can be configured fairly easily when the database schema is arranged into a clear hierarchy where each ISA-95 type and its properties are arranged into a table in rows and columns, respectively. This is done using the `createISA95MaterialHierarchy` function. However, it has a many parameters, which need to be carefully chosen. A detailed example of choosing the correct parameters is provided in Section 6.2.1.

6.1.3 Platform Behavior

The node manager begins the mapping process by reading the URI of devices from the configuration and creating clients to connect to them. Folders are created on the aggregating server to contain and separate nodes from different devices. Clients, each created to access a device, access the objects folder of the device it has connected to, and begin to follow references in the address spaces starting from the Objects folder. The scan is a depth-first search, using the Browse service to find references between nodes. Since an OPC UA address space can contain cycles, visited nodes are added to a hash set, which is consulted to detect and avoid cycles during the scan. Each node and its references are duplicated into the aggregating address space as a proxy node. As the contents of the aggregated servers are duplicated, the client connecting to the aggregated server, as well as the remote NodeId, are added to the remote node mapping. A list of NodeIds to be historized is read, and HDA mappings are made to them. Historized nodes' database identifiers are requested from the database, and calls for historical data for that node are mapped to a database call with that identifier. A subscription is placed on historized nodes, and changed values are set to be added to the database with the same identifier. After the address spaces are duplicated, other parts of the configuration, alarms and mappings to a database, are executed.

6.2 Applications

To demonstrate the operation of the implementation detailed in Section 6.1, the application was configured to execute use cases, suggested by the main partner in the project. The use cases were selected to demonstrate applications of OPC UA aggregating servers that would be of use in real-world FMS.

6.2.1 Tool Information Representation

The aggregating server was configured to represent tool information in a database according to ISA-95 for OPC UA[25]. The represented information was added to the database used in the development environment. Tables in a

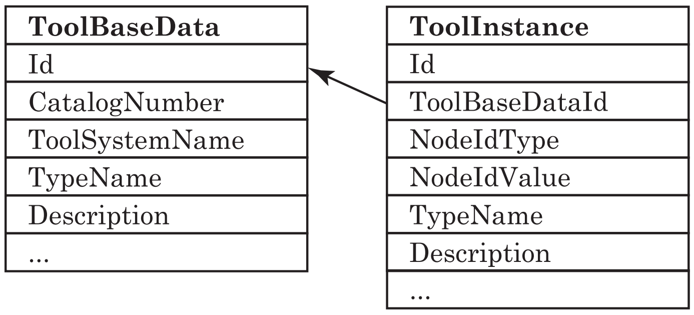
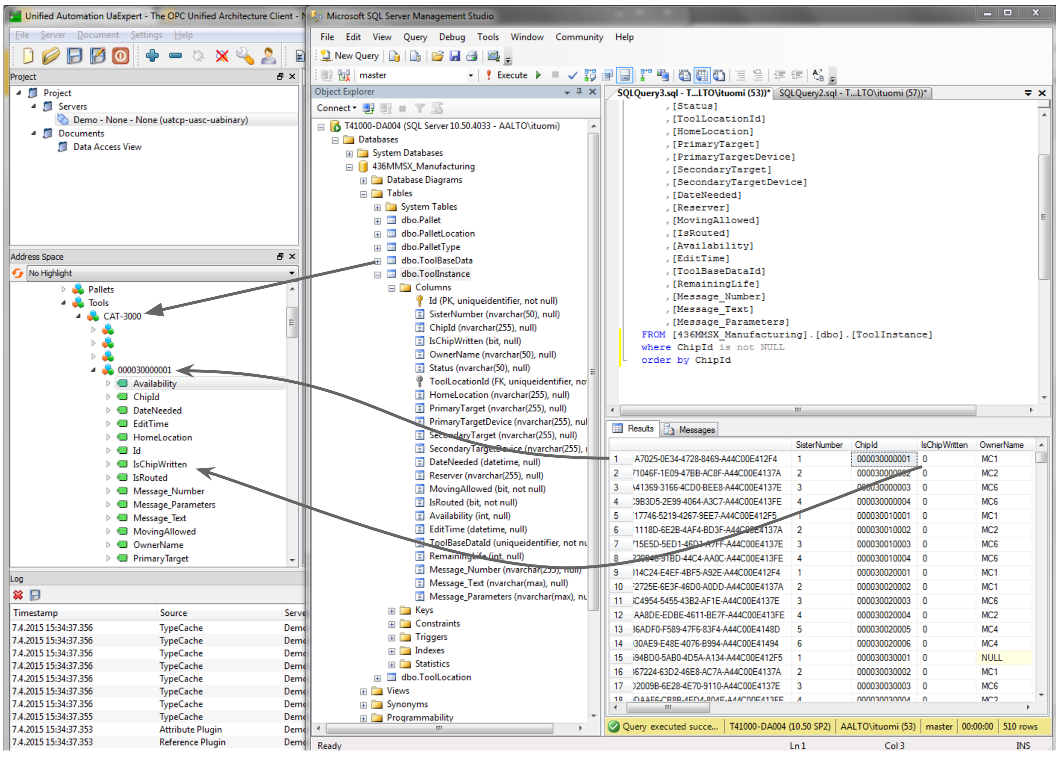


Figure 6.2: The part of the FMS controller database schema containing mapped material information.



```
Project
  Servers
    Demo - None - None (uatcp-uasc-uabinary)
  Documents
    Data Access View
```

```
Pallets
Tools
  CAT-3000
    000030000001
      Availability
      ChipId
      DateNeeded
      EditTime
      HomeLocation
      Id
      IsChipWritten
      IsRouted
      Message_Number
      Message_Parameters
      Message_Text
      MovingAllowed
      OwnerName
      PrimaryTarget
```

Timestamp	Source	Serve
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.356	TypeCache	Dem
7.4.2015 15:34:37.355	TypeCache	Dem
7.4.2015 15:34:37.353	Attribute Plugin	Dem
7.4.2015 15:34:37.353	Reference Plugin	Dem

```
T41000-DA004 (SQL Server 10.50.4033 - AALTO\ituomi)
  Databases
    System Databases
    436MMSX_Manufacturing
      Database Diagrams
      Tables
        System Tables
        dbo.Pallet
        dbo.PalletLocation
        dbo.PalletType
        dbo.ToolBaseData
        dbo.ToolInstance
          Columns
            Id (PK, uniqueidentifier, not null)
            SisterNumber (nvarchar(50), null)
            ChipId (nvarchar(255), null)
            IsChipWritten (bit, null)
            OwnerName (nvarchar(50), null)
            Status (nvarchar(50), null)
            ToolLocationId (FK, uniqueidentifier, no
            HomeLocation (nvarchar(255), null)
            PrimaryTarget (nvarchar(255), null)
            PrimaryTargetDevice (nvarchar(255), nul
            SecondaryTarget (nvarchar(255), null)
            SecondaryTargetDevice (nvarchar(255), 
            DateNeeded (datetime, null)
            Reserver (nvarchar(255), null)
            MovingAllowed (bit, not null)
            IsRouted (bit, not null)
            Availability (int, null)
            EditTime (datetime, null)
            ToolBaseDataId (uniqueidentifier, not nu
            RemainingLife (int, null)
            Message_Number (nvarchar(255), null)
            Message_Text (nvarchar(max), null)
            Message_Parameters (nvarchar(max), nu
          Keys
          Constraints
          Triggers
          Indexes
          Statistics
        dbo.ToolLocation
      Views
      Synonyms
      Programmability
```

```sql
      ,[Status]
      ,[ToolLocationId]
      ,[HomeLocation]
      ,[PrimaryTarget]
      ,[PrimaryTargetDevice]
      ,[SecondaryTarget]
      ,[SecondaryTargetDevice]
      ,[DateNeeded]
      ,[Reserver]
      ,[MovingAllowed]
      ,[IsRouted]
      ,[Availability]
      ,[EditTime]
      ,[ToolBaseDataId]
      ,[RemainingLife]
      ,[Message_Number]
      ,[Message_Text]
      ,[Message_Parameters]
  FROM [436MMSX_Manufacturing].[dbo].[ToolInstance]
  where ChipId is not NULL
  order by ChipId
```

		SisterNumber	ChipId	IsChipWritten	OwnerName
1	A7025-0E34-4728-8469-A44C00E412F4	1	000030000001	0	MC1
2	71046F-1E09-47BB-AC8F-A44C00E4137A	2	000030000002	0	MC2
3	41369-3166-4CD0-BEE8-A44C00E4137E	3	000030000003	0	MC6
4	C9B3D5-2E99-4064-A3C7-A44C00E413FE	4	000030000004	0	MC6
5	17746-5219-4267-9EE7-A44C00E412F5	1	000030010001	0	MC1
6	1118D-6E2B-4AF4-BD3F-A44C00E4137A	2	000030010002	0	MC2
7	715E5D-5ED1-46D1-A7FF-A44C00E4137E	3	000030010003	0	MC6
8	220046-91BD-44C4-AA0C-A44C00E413FE	4	000030010004	0	MC6
9	014C24-E4EF-4BF5-A92E-A44C00E412F4	1	000030020001	0	MC1
10	F2725E-6E3F-46D0-A0DD-A44C00E4137A	2	000030020002	0	MC1
11	C4954-5455-43B2-AF1E-A44C00E4137E	3	000030020003	0	MC6
12	AA8DE-EDBE-4611-BE7F-A44C00E413FE	4	000030020004	0	MC2
13	36ADF0-F589-47F6-83F4-A44C00E4148D	5	000030020005	0	MC4
14	30AE9-E48E-4076-B994-A44C00E41494	6	000030020006	0	MC4
15	594BD0-5AB0-4D5A-A134-A44C00E412F5	1	000030030001	0	NULL
16	367224-63D2-46E8-AC7A-A44C00E4137A	2	000030030002	0	MC1
17	02009B-6E28-4E70-9110-A44C00E4137E	3	000030030003	0	MC6

Query executed succe... | T41000-DA004 (10.50 SP2) | AALTO\ituomi (53) | master | 00:00:00 | 510 rows

Figure 6.3: UAExpert view of Mapping Tool information from the database as ISA-95 Material information. Arrows show connections between the database and the OPC UA address space.

relational database used by a real-world FMS were duplicated and populated with test data generated by a development tool used by the FMS developer. The part of the schema containing tool information, illustrated in Figure 6.2, was found to contain a suitable hierarchy to be mapped to the ISA-95 specification. The aggregating OPC UA server was configured to map the tables into its address space according to ISA-95 Material Information. An object node of the ISA-95 type MaterialClass called *Tools* was created to represent and organize all tool material information. The configuration parameters specify the table ToolBaseData as the source of Material Definitions, and ToolInstance as the source of Material Lots. Additionally, the field used for OPC UA node names (*"CatalogNumber"* and *"ChipId"*), the fields containing primary keys for the tables (*"Id"*), and the field containing the foreign key for the table containing material lots *("ToolBaseDataId")* were given as parameters. This configuration is enough to execute the mapping.

As the database data is guaranteed to comply with a certain schema, so there were no complications in mapping the contents of the database into the address space. The resulting address space structure presented the database material information correctly. Writing values to the database via the address space mapping also worked without complications. The result of mapping the database tables, as well as a view of the database tables themselves in Microsoft SQL Management Studio are shown in Figure 6.3.

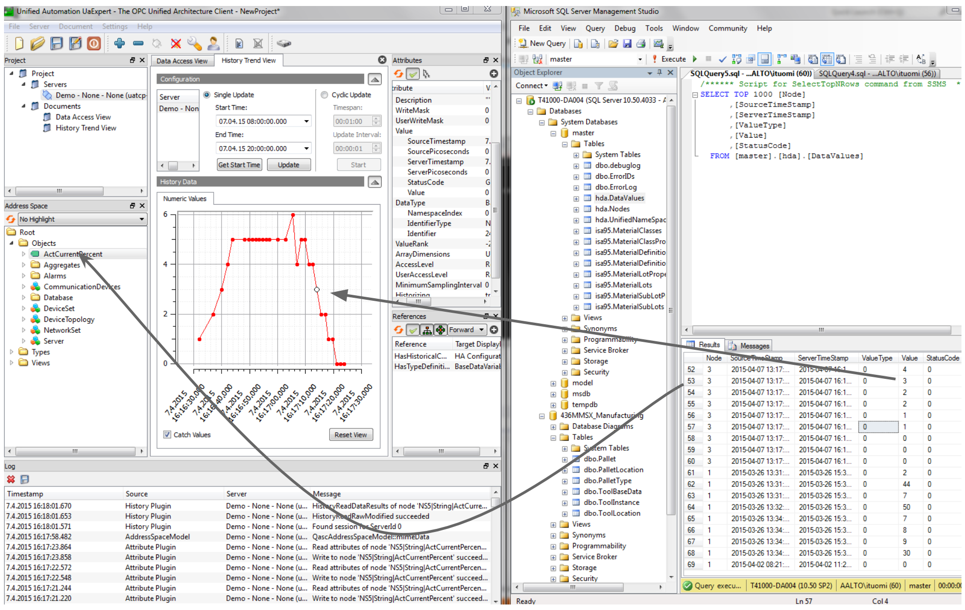
6.2.2 Power Consumption Monitoring

Changes in the power consumption of moving equipment can indicate a need for maintenance. In this use case, the aggregating server is configured to measure power consumption when lifting a test weight. However, the aggregated OPC UA servers were created for testing purposes, and do not simulate actual devices in any way. Measurement of power consumption was illustrated by creating and historizing an OPC UA node and changing its value manually in order for the time series to be saved in database. The added values were

The view of reading the historized values afterwards from the database can be seen in Figure 6.4.

6.2.3 Detection of Inconsistent Data

A photocell sensor, accessible via PLC, provides sensor data indicating whether a pallet is in the loading bay. Simultaneously, a FMS controller database in-



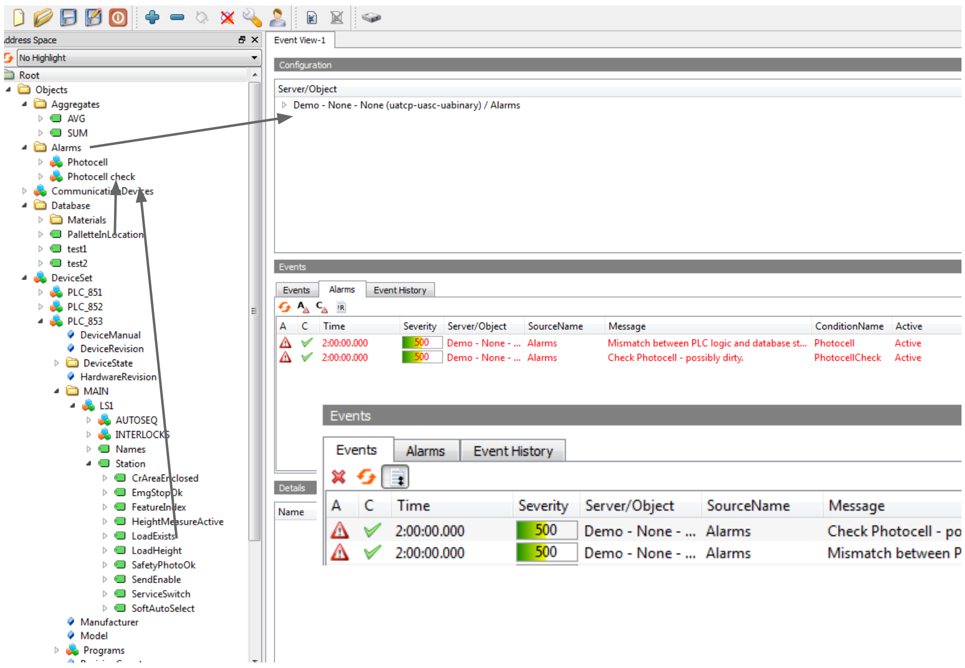
```
/****** Script for SelectTopNRows command from SSMS  *
SELECT TOP 1000 [Node]
      ,[SourceTimeStamp]
      ,[ServerTimeStamp]
      ,[ValueType]
      ,[Value]
      ,[StatusCode]
  FROM [master].[hda].[DataValues]
```

	Node	SourceTimeStamp	ServerTimeStamp	ValueType	Value	StatusCode
52	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	4	0
53	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	3	0
54	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	2	0
55	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	2	0
56	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	1	0
57	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	1	0
58	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	0	0
59	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	0	0
60	3	2015-04-07 13:17:...	2015-04-07 16:1...	0	0	0
61	1	2015-03-26 13:31:...	2015-03-26 15:3...	0	2	0
62	1	2015-03-26 13:31:...	2015-03-26 15:3...	0	44	0
63	1	2015-03-26 13:31:...	2015-03-26 15:3...	0	7	0
64	1	2015-03-26 13:32:...	2015-03-26 15:3...	0	50	0
65	1	2015-03-26 13:34:...	2015-03-26 15:3...	0	7	0
66	1	2015-03-26 13:34:...	2015-03-26 15:3...	0	8	0
67	1	2015-03-26 13:34:...	2015-03-26 15:3...	0	9	0
68	1	2015-03-26 13:34:...	2015-03-26 15:3...	0	30	0
69	1	2015-04-02 08:21:...	2015-04-02 11:2...	0	0	0

```
Timestamp               Source              Server                    Message
7.4.2015 16:18:01.670   History Plugin      Demo - None - None (u...  HistoryReadDataResults of node 'NS5|String|ActCurre...
7.4.2015 16:18:01.653   History Plugin      Demo - None - None (u...  HistoryReadRawModified succeeded
7.4.2015 16:18:01.571   History Plugin      Demo - None - None (u...  Found session for ServerId 0
7.4.2015 16:17:58.482   AddressSpaceModel   Demo - None - None (u...  QascAddressSpaceModel::mimeData
7.4.2015 16:17:23.864   Attribute Plugin    Demo - None - None (u...  Read attributes of node 'NS5|String|ActCurrentPercen...
7.4.2015 16:17:23.858   Attribute Plugin    Demo - None - None (u...  Write to node 'NS5|String|ActCurrentPercent' succeed...
7.4.2015 16:17:22.572   Attribute Plugin    Demo - None - None (u...  Read attributes of node 'NS5|String|ActCurrentPercen...
7.4.2015 16:17:22.548   Attribute Plugin    Demo - None - None (u...  Write to node 'NS5|String|ActCurrentPercent' succeed...
7.4.2015 16:17:21.244   Attribute Plugin    Demo - None - None (u...  Read attributes of node 'NS5|String|ActCurrentPercen...
7.4.2015 16:17:21.220   Attribute Plugin    Demo - None - None (u...  Write to node 'NS5|String|ActCurrentPercent' succeed...
```

Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*
Microsoft SQL Server Management Studio
Data Access View
History Trend View
Single Update
Cyclic Update
Start Time: 07.04.15 08:00:00.000
End Time: 07.04.15 20:00:00.000
Get Start Time
Update
Numeric Values
Catch Values
Reset View
SQLQuery5.sql - ...ALTO\ituomi (60))
SQLQuery4.sql - ...ALTO\ituomi (56))
Results
Messages
Query execu...
T41000-DA004 (10.50 SP2)
AALTO\ituomi (60)
master
Ln 57
Col 4

Figure 6.4: UAExpert view of reading historical data. Values in the table on the right are displayed in a graph on the left.



Server/Object
Demo - None - None (uatcp-uasc-uabinary) / Alarms

A	C	Time	Severity	Server/Object	SourceName	Message	ConditionName	Active
⚠	✓	2:00:00.000	500	Demo - None - ...	Alarms	Mismatch between PLC logic and database st...	Photocell	Active
⚠	✓	2:00:00.000	500	Demo - None - ...	Alarms	Check Photocell - possibly dirty.	PhotocellCheck	Active

A	C	Time	Severity	Server/Object	SourceName	Message
⚠	✓	2:00:00.000	500	Demo - None - ...	Alarms	Check Photocell - po
⚠	✓	2:00:00.000	500	Demo - None - ...	Alarms	Mismatch between P

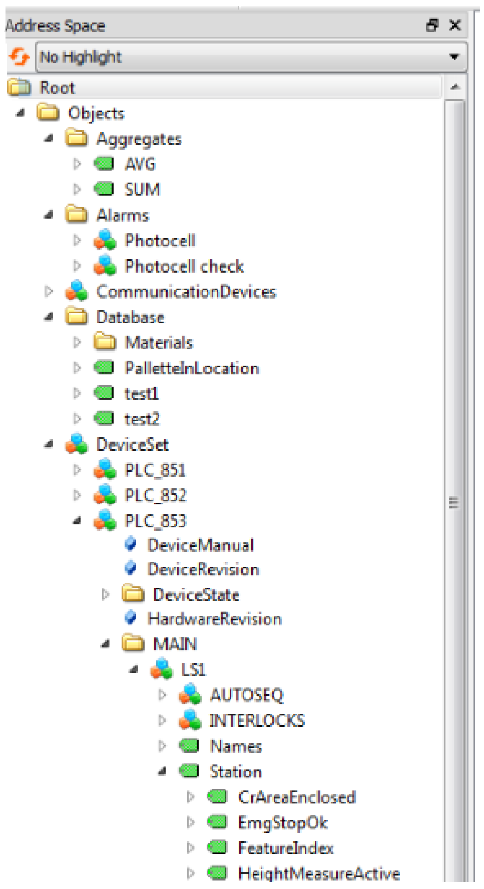
Figure 6.5: UAExpert view of activated alarms which indicate a data inconsistency.

dicates whether a planned schedule indicates a pallet is in the bay. When these values do not agree it can be concluded that there is an error or inconsitency in the FMS. Possible causes are that the sensor is broken or dirty, the pallet is too reflective, the user has manually loaded the pallet, or that there is some other unknown reason. Such an observation typically requires production to be stopped in order to avoid further problems. An alarm was created that checks the consistency of the value in the two values in question. When the values do not agree, the alarm activates and sends an alarm event. Figure 6.5 shows the nodes read to execute the alarm condition, and the alarm event recieved by the subscribed UAExpert client. The aggregated servers and database created for experimentation do not simulate the functioning of a FMS. Therefore although alarms checked actual database and PLC values, their values were modified manually during experimentation for the alarm condition to be fulfilled, causing the alarm event to be raised.

6.2.4 Resulting Address Space Structure

The address space resulting from the configuration is visible in Figure 6.6. The Database folder contains a node mapped to the PalletteInLocation procedure in the database. The variables *test1* and *test2* are mapped to database procedures simply returning a constant value, and the variables *AVG* and *SUM* are mapped to an aggregating function combining both test variables into their average, and sum, respectively.

The object nodes under *DeviceSet*, PLC_851, PLC_852 and PLC_853 are address spaces of aggregated devices. The folder *MAIN*, and the nodes it organizes are generated according to data structures declared in PLC code.



```
Address Space
No Highlight
Root
  Objects
    Aggregates
      AVG
      SUM
    Alarms
      Photocell
      Photocell check
    CommunicationDevices
    Database
      Materials
      PalletteInLocation
      test1
      test2
    DeviceSet
      PLC_851
      PLC_852
      PLC_853
        DeviceManual
        DeviceRevision
        DeviceState
        HardwareRevision
        MAIN
          LS1
            AUTOSEQ
            INTERLOCKS
            Names
            Station
              CrAreaEnclosed
              EmgStopOk
              FeatureIndex
              HeightMeasureActive
```

Figure 6.6: View of resulting address space instance in UAExpert.

Chapter 7

Conclusions

The goal of this thesis was to define and demonstrate aggregating OPC UA servers as part of a FMS. The requirements of an aggregating OPC UA server for FMS were defined based on previous work on aggregating servers, as well as work on the specific needs of FMS. A software design fulfilling the requirements was detailed and partially implemented. The resulting implementation was used to develop applications to comply with certain scenarios to demonstrate its functionality.

The requirements were defined in Chapter 4. The minimum requirements of an aggregating server architecture include identifying and connecting to devices and providing clients access to them. Observation of the FMS requires notifying operators in user-defined circumstances. Therefore user-configurable alarms are required. Historical data has many important uses for FMS, including time-series analysis and improving the traceability of production, and support for it was found to be an important requirement for the aggregating server. As FMS differ, the application logic required to effectively operate them also vary. Therefore the possibility of creating connections between the address space and custom functionality for varying use cases is required. Mapping database contents to the address space was found to be a specific requirement in the context of this work.

The software design for an aggregating server for FMS fulfilling the requirements was described in Chapter 5. Connectivity is done with a standard OPC UA server and clients as part of the aggregating server. The address space is organized with folders dividing the address space according to node usage. Nodes on the aggregating server are connected to devices' address spaces with mappings pairing the aggregating server nodes to the information needed to access their data source. Parts of device address space contents were scanned, duplicated into the aggregating server's address space, and mapped back to the devices. It was not evident that more complicated transformations than directly duplicating the devices' address space to the aggregating server would be useful, since further transformations can be done within the aggregating server. Simple one-to-one mapping also has the benefits of simplicity and transparency. Some nodes were also mapped to database procedures. Database tables and procedures were mapped to OPC UA HDA functionality, enabling saving and persisting node values. Alarms are managed by an alarm manager subscribing to values alarm states depend on, and executing alarm conditions when subscribed values change. This enables alarm conditions to be calculated only when needed.

The implementation of the designed aggregating server and experimentation on it was undertaken in Chapter 6. In practice the implementation is an application development platform featuring basic functionality with minimal configuration, with the ability to be extended freely by programming it. Configuration of the application platform to the selected use cases was straight-forward, and required little programming. In general, the implementation of the an aggregating server was found to be practical, and the end result fulfilled the requirements. It seems likely that the aggregating OPC UA server as designed can be configured to facilitate many use cases in FMS. Although the experimentation was conducted in a controlled environment, there are no obvious reasons that would invalidate the design for comparable applications in real-world use.

This thesis finds no objections to the use of OPC UA as a communication protocol for FMS. In the scope of this work, OPC UA was found to be a practical option, at least for communication to and from devices, provided they feature OPC UA support. The benefits of using OPC UA increase greatly when compatible equipment is available and does not need to be specifically integrated. However, the extent to which it is beneficial to use OPC UA as a communication protocol is unclear when considering it for use in communications between different systems within a company.

Additional work needs to be done to make the aggregating OPC UA server ready for production use. Configuration should be possible to at least some extent without recompiling the entire application, preferably through text-based configuration. Subscription to database values should be possible in a way that the database itself informs the aggregating server of changed values. In the scope of work done in this thesis, namespace conflicts were not an issue. However, they are likely to become an issue in a production environment. A robust solution should be developed to avoid namespace conflicts.

Whether or not specialized OPC UA types for aggregating servers as suggested by Großmann et al. [4] would be useful is a possible topic for further research. Also formalizing mapping rules, and codifying them into the OPC UA address space, as per Großmann [4], might be an interesting direction, but not an absolute necessity. These directions needs to be carefully considered, as user-defined OPC UA type definitions do not provide direct benefit in themselves, but must be specifically utilized by clients according to some pre-existing agreement or convention. No immediately obvious applications for clients utilizing these types are known at the time of writing this thesis, but they may exist.

References

[1] ASIKAINEN, J. OPC UA Java History Gateway with Inherent Database Integration. Master's thesis, Department of Automation and Systems Technology, Aalto University School of Electrical Engineering, Espoo, Finland, 2010.

[2] BECKHOFF AUTOMATION. TwinCAT OPC UA Server. `http://www.beckhoff.fi/english.asp?twincat/twincat_opc_ua_server.html`. [Online; accessed 2015, April 8.].

[3] BROWNE, J., DUBOIS, D., RATHMILL, K., SETHI, S. P., AND STECKE, K. E. Classification of flexible manufacturing systems. *The FMS magazine 2*, 2 (1984), 114–117.

[4] GROSSMANN, D., BREGULLA, M., BANERJEE, S., SCHULZ, D., AND BRAUN, R. OPC UA server aggregation—The foundation for an internet of portals. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE* (2014), IEEE, pp. 1–6.

[5] GUPTA, Y. P., AND GOYAL, S. Flexibility of manufacturing systems: concepts and measurements. *European journal of operational research 43*, 2 (1989), 119–135.

[6] HASTBACKA, D., BARNA, L., KARAILA, M., LIANG, Y., TUOMINEN, P., AND KUIKKA, S. Device status information service architecture for condition monitoring using OPC UA. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE* (2014), IEEE, pp. 1–7.

[7] INTERNATIONAL ELECTROTECHNICAL COMMISSION. IEC 61131-3. *Programmable Controllers - Part 3* (1993).

[8] INTERNATIONAL SOCIETY OF AUTOMATION. ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod) Enterprise-Control System Integration.

[9] Kuisma, V. M., et al. *Joustavan konepaja-automaation käyttöönoton onnistumisen edellytykset.* VTT Technical Research Centre of Finland, 2007.

[10] Laukkanen, E. Java source code generation from OPC UA information models. Master's thesis, Department of Automation and Systems Technology, Aalto University School of Electrical Engineering, Espoo, Finland, 2013.

[11] Leitner, S.-H., and Mahnke, W. OPC UA–service-oriented architecture for industrial applications. *ABB Corporate Research Center* (2006).

[12] Mahnke, W., Leitner, S.-H., and Damm, M. *OPC Unified Architecture*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[13] Melander, L. Interface Integration Challenges in Future Flexible Manufacturing Systems. Master's thesis, Faculty of Engineering Sciences, Tampere University Of Technology, 2015.

[14] OPC Foundation. OPC Alarms & Events (OPC AE).

[15] OPC Foundation. OPC Data Access (OPC DA).

[16] OPC Foundation. OPC Historical Data Access (OPC HDA).

[17] OPC Foundation. OPC Unified Architecture Specification Part 1: Overview and Concepts, 2012.

[18] OPC Foundation. OPC Unified Architecture Specification Part 11: Historical Access, 2012.

[19] OPC Foundation. OPC Unified Architecture Specification Part 2: Security Model, 2012.

[20] OPC Foundation. OPC Unified Architecture Specification Part 3: Address Space Model, 2012.

[21] OPC Foundation. OPC Unified Architecture Specification Part 4: Services, 2012.

[22] OPC Foundation. OPC Unified Architecture Specification Part 6: Mappings, 2012.

[23] OPC Foundation. OPC Unified Architecture Specification Part 9: Alarms & Conditions, 2012.

[24] OPC Foundation. OPC Unified Architecture For Devices, 2013.

[25] OPC Foundation. OPC Unified Architecture For ISA-95 Common Object Model, 2013.

[26] Palonen, O. Object-oriented implementation of OPC UA information models in java. Master's thesis, Information and Computer Systems in Automation, Aalto University School of Science and Technology, Espoo, Finland, 2010.

[27] Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. A design science research methodology for information systems research. *Journal of management information systems 24*, 3 (2007), 45–77.

[28] PLCOpen and OPC Foundation. OPC Unified Architecture Information Model for IEC 61131-3, 2010.

[29] Prosys. OPC UA Java SDK. `https://www.prosysopc.com/products/opc-ua-java-sdk/`. Accessed: 2015-04-05.

[30] Prosys PMS Ltd. OPC UA Historian. `http://www.prosysopc.com/products/opc-ua-historian/`. [Online; accessed 2015, May 11.].

[31] Shivanand, H. K. *Flexible Manufacturing System.* New Age International, 2006.

[32] Unified Automation. C Based OPC UA Server SDK. `https://www.unified-automation.com/products/server-sdk/ansi-c-ua-server-sdk.html`. Accessed: 2015-04-05.

[33] Unified Automation. C++ Based OPC UA Server SDK. `https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html`. Accessed: 2015-04-05.

[34] Unified Automation. Java Based OPC UA Server SDK. `https://www.unified-automation.com/products/server-sdk/java-ua-server-sdk.html`. Accessed: 2015-04-05.

[35] Unified Automation. .NET Based OPC UA Client & Server SDK (Bundle). `https://www.unified-automation.com/products/server-sdk/net-ua-server-sdk.html`. Accessed: 2015-04-05.

[36] Unified Automation GmbH. UaGateway. `https://www.unified-automation.com/products/wrapper-and-proxy/uagateway.html`. [Online; accessed 2015, May 11.].

[37] von Alan, R. H., March, S. T., Park, J., and Ram, S. Design science in information systems research. *MIS quarterly 28*, 1 (2004), 75–105.