An Aggregated Integration Platform - an approach for the aggregation of Information Models

Dissertation
zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

von **M.Sc. Suprateek Banerjee**
geb am 16.12.1986 in Azamgarh, Indien

genehmigt durch die Fakultät für Elektrotechnik und Informationstechnik
der Otto-von-Guericke-Universität Magdeburg

Gutachter:

1. Prof. Dr.-Ing. Christian Diedrich
2. Prof. Dr.-Ing. habil. Leon Urbas
3. Prof. Dr.-Ing. Daniel Großmann

Promotionskolloquium am 02. November 2023

Acknowledgement

This page is only available in the print version due to the regulations of the Otto-von-Guericke University.

Abstract

In today's automation landscape, a clear trend towards distribution can be noticed. Starting from processing power to where data about the automation systems are being stored and used are being decentralized. The devices at the shop floor are getting smaller and "smarter", in the sense that they are now becoming capable of describing their own health or energy related metrics, which are of relevance for higher level monitoring systems, for example. Cloud based systems can now access data from all over the shop floor and apply machine learning algorithms to realize use-cases like predictive and preventive maintenance. However, as one can realize, the first step in this scenario is to get the data in a uniform and standardized manner. This is where information models come into the picture. Standardized use-case specific information models can help realize the true potential of Industry 4.0. There is indeed a plethora of standardized information models within the automation domain, looking at different aspects of the same automation system. Therefore, an obvious need to harmonize information models and to minimize any overlaps between them arises. This is seen from the point of view of an aggregating application that wants to collect information based on standardized information models. This thesis looks at some of state-of-the-art information models and exemplifies a scenario to explain the concept. It collects the requirements for the creation of a so-called aggregated information model and an aggregated integration platform respectively. Based on the requirements, this thesis proposes an architecture of such an aggregated integration platform, that aggregates several information models while considering that they might have overlaps. It also proposes a solution to solve these overlaps and uses this solution to create the aforementioned aggregated information model.

Zusammenfassung

In der heutigen Automatisierungslandschaft ist ein deutlicher Trend zur Dezentralisierung zu erkennen. Angefangen bei der Verarbeitung bis hin zur Speicherung von Daten und Informationen – das findet dezentral ("on the edge") statt, verteilt über das System. Die Geräte in der Fertigung werden kleiner und "intelligenter" in dem Sinne, dass sie nun in der Lage sind, ihre eigenen zustands- oder energiebezogenen Metriken zu beschreiben, die z. B. für übergeordnete Überwachungssysteme relevant sind. Cloud-basierte Systeme können nun auf Daten aus dem gesamten Betrieb zugreifen und Algorithmen des maschinellen Lernens anwenden, um Anwendungsfälle wie vorausschauende und vorbeugende Wartung zu realisieren. Der erste Schritt in diesem Szenario besteht jedoch darin, die Daten in einer einheitlichen und standardisierten Weise zu erfassen und zur Verfügung zu stellen. Und hier kommen Informationsmodelle ins Spiel. Standardisierte, anwendungsspezifische Informationsmodelle können dabei helfen, das wahre Potenzial von Industrie 4.0 auszuschöpfen. Es gibt in der Tat eine Fülle von standardisierten Informationsmodellen im Bereich der Automatisierung, die (jedes Modell für sich) unterschiedliche Aspekte desselben Automatisierungssystems betrachten. Die Notwendigkeit, Informationsmodelle zu harmonisieren und die Überschneidungen zwischen ihnen zu minimieren, ist offensichtlich. Die Arbeit betrachtet dies aus Sicht einer aggregierenden Anwendung, die Informationen auf der Grundlage standardisierter Informationsmodelle erfasst. Sie betrachtet einige der gängigen Informationsmodelle , stellt ein Szenario zur Erläuterung des Konzepts exemplarisch dar und gestellt und trägt die Anforderungen für die Erstellung eines sogenannten aggregierten Informationsmodells bzw. einer aggregierten Integrationsplattform zusammen. Basierend auf den Anforderungen wird in dieser Arbeit eine Architektur für eine solche aggregierte Integrationsplattform vorgeschlagen, die mehrere Informationsmodelle aggregiert und dabei berücksichtigt, dass diese Überschneidungen aufweisen können. Sie schlägt auch eine Lösung vor, wie mit diesen Überschneidungen umzugehen ist und verwendet diese Lösung, um das oben erwähnte aggregierte Informationsmodell zu erstellen.

Contents

Acknowledgement **iii**

Abstract **v**

Zusammenfassung **vii**

1 Introduction **1**

1.1 Motivation and Problem Statement 1
1.2 Goal of the thesis 2
1.3 Structure of the thesis 3

2 Management of Cyber-physical Production Systems **7**

2.1 Complex Automation Systems 7
2.1.1 Types of Automation Systems 8
2.1.2 Structure of Automation Systems 9
2.1.3 Life Cycle of Automation Systems 11
2.2 Cyber Physical Production Systems 14
2.2.1 Structure of a CPPS 15
2.2.2 Information Modeling in a CPPS 16
2.3 Information Integration in a CPPS 17
2.3.1 The resulting connection mesh 18
2.3.2 Enterprise Application Integration 19
2.3.3 Aggregated Information Model 21
2.4 Summary and Task Definition 24

3 Methods and Tools for Information Integration **27**

3.1 Development of Information Integration 28
3.1.1 The General Information Integration Process 28
3.2 The Information Modeling Process 32
3.2.1 Conceptualization of Information Models 33
3.2.2 Modeling Methodologies 35

3.2.3 Modeling Steps . . . 37
3.3 A Common Landscape of Information Models in Automation . . . 40
3.3.1 Reference Models . . . 41
3.3.2 Model Universals . . . 41
3.3.3 Concept for consistent use of Engineering Models in Automation . . . 42
3.3.4 SemAnz4.0 . . . 42
3.3.5 RAMI 4.0 . . . 44
3.4 OPC UA - An Enabler for Industry 4.0 . . . 46
3.4.1 Motivation and Overview . . . 46
3.4.2 OPC UA Base Layer . . . 47
3.4.3 OPC UA Model Extensions . . . 51
3.5 Summary . . . 53

4 Solution Concept for the Aggregated Integration Platform 55
4.1 Basic Concept . . . 56
4.1.1 Overview of the concept idea . . . 56
4.1.2 Method for the solution development . . . 58
4.2 Analysis . . . 59
4.2.1 Actors and Use-Cases . . . 60
4.2.2 Requirements for the Aggregated Integration Platform . . . 63
4.2.3 Entities in an Aggregated Information Model . . . 65
4.2.4 An Aggregation Scenario from the Process Industry . . . 69
4.2.5 Considerations for Information Aggregation . . . 71
4.2.6 Merging of Information Models . . . 73
4.2.7 Demonstration of a Merged Information Model with Graphs 80
4.3 Base Model of Information Aggregation . . . 93
4.3.1 Aggregated Information Model . . . 93
4.3.2 Aggregation Communication Model . . . 95
4.3.3 Consumer Operation Model . . . 97
4.4 Meta-model of Information Aggregation . . . 99
4.4.1 Basic Structure of the Meta-model . . . 100
4.4.2 Data and State Model . . . 102
4.4.3 Basic Methods . . . 103
4.4.4 Addressing Information . . . 104
4.4.5 Communication Sequences . . . 105
4.4.6 Consumer Business Logic . . . 105
4.4.7 Graphical User Interface . . . 106
4.5 Summary . . . 107

5 Architecture of the Aggregated Integration Concept 109
5.1 Architecture Concept . . . 110
5.1.1 Overview of the Architecture Concept . . . 110
5.1.2 Procedure for the Architecture development . . . 113
5.2 Structural Model . . . 114
5.2.1 Conceptual Class Diagrams of the platform modules . . . 114
5.3 Definition of Interfaces . . . 122
5.3.1 Login-Interface . . . 124
5.3.2 AIM-Interface . . . 124
5.3.3 COM-Interface . . . 125
5.3.4 Mapping-Interface . . . 125
5.3.5 ACM-Interface . . . 126
5.3.6 Configuration-Interface . . . 126
5.4 Behavioral Model . . . 127
5.4.1 Read Access to a UIM element . . . 127
5.4.2 Write Access to a UIM element . . . 129
5.4.3 Subscribing to a UIM instance . . . 130
5.4.4 Addition of a UIM element by Consumer . . . 132
5.4.5 Deletion of a UIM element by Consumer . . . 133
5.4.6 Dynamic Addition of a UIM element . . . 135
5.4.7 Dynamic Deletion of a UIM element . . . 136
5.5 Summary . . . 137

6 Realization and Review 139
6.1 Overview of the Implementation . . . 139
6.1.1 Overview of the Prototype . . . 140
6.1.2 Process of Prototype Implementation . . . 141
6.2 Implementation Technologies . . . 142
6.2.1 Description of Mapping Rules . . . 143
6.2.2 Client-Server Technology . . . 143
6.3 Illustration of the aggregated platform . . . 143
6.3.1 Underlying Servers . . . 144
6.3.2 Aggregation Server . . . 147
6.3.3 Configuration . . . 150
6.3.4 Consumer Client . . . 150
6.3.5 Mapping Rules Repository . . . 151
6.4 Evaluation of the implemented prototype . . . 155
6.5 Summary . . . 156

7 Conclusion and Outlook **157**
7.1 Conclusion 157
7.2 Outlook 159

Appendices **161**

Annex A **163**

References **183**

List of Publications **191**

Curriculum Vitae **193**

Declaration of Honor **195**

Chapter 1

Introduction

With the advent of Industry 4.0, the rapid growth in processing power of devices and at the same time the reduction in their sizes, effective and efficient information modeling is becoming ever more important. Today, the components being used in the automation industry are already capable of exposing information about themselves in the form of well defined information models making them even more capable and flexible. This however makes managing all the information highly complex and with so many information sources and sinks, it is increasingly becoming a challenge to manage it effectively.

1.1 Motivation and Problem Statement

In today's automation scenario, with the increasing trend towards distributed systems, the information models being exposed by those systems are gaining importance. With devices becoming more and more intelligent in terms of their processing power capabilities, it is now possible for the higher levels of the automation pyramid (e.g. Manufacturing Execution Systems) to access information directly from the field devices. Figure 1.1 shows the gradual phase-in of the well known automation pyramid into a system where there is seamless communication between entities from different levels of the traditional automation pyramid. This not only means vertical communication but also horizontal communication between entities on the same level e.g direct communication between a robot and a machine vision system. An advantage of this development is that devices or subsystems are already individually accessible at a stage in the life cycle of the automation system when the system as a whole has not yet been engineered or commissioned ([Gro+14b]). Although this increases the flexibility of the entire automation system manifold, it brings in the challenges of a fully-meshed network of information sources and sinks.

In order to solve this aforementioned connection mesh, there is need for a solution that aggregates the various information being exposed by the information sources. However, to effectively aggregate information from several sources into a single information source, the overlaps (if any) between the existing information need to be solved. Upon effectively aggregating information from several sources, consumers can retrieve information from the aggregated information source depending upon their use case i.e. applications looking for information related to use-cases such as Asset Management or Condition Monitoring can access this use-case specific information from the aggregated source.

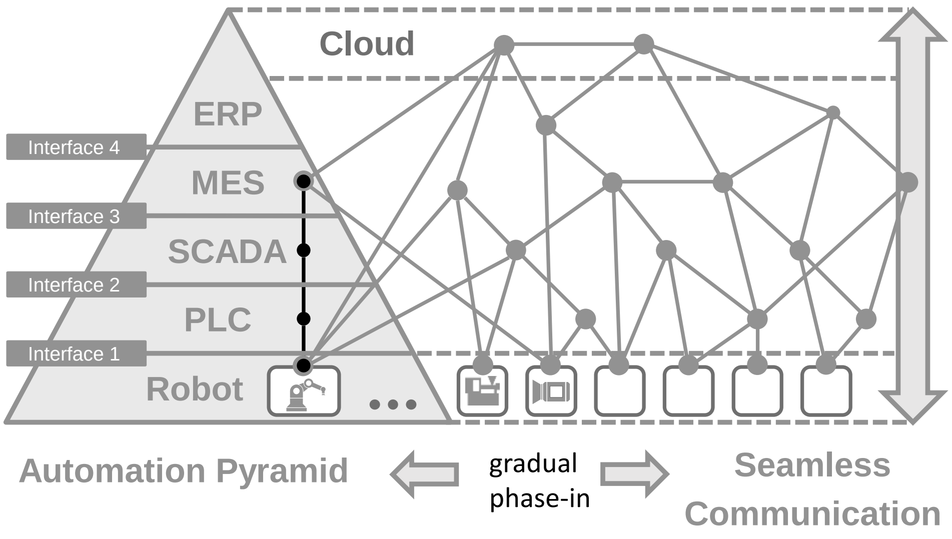


Figure 1.1: The gradual phase-in of the automation pyramid

1.2 Goal of the thesis

A similar problem (as that mentioned in the previous section) has already been tackled by operational information systems i.e. the challenges posed by the multiplicity of couplings resulting from the connection mesh formed by the multitude of applications interacting with each other. As a solution to this problem the concept of *Enterprise Application Integration* could be used to reduce the number of couplings. In this concept various consumer applications connect to a so called *Integration Platform* and not directly to the information sources.
Similarly, with respect to the scenario mentioned in the previous section, an *Ag-*

gregated Integration Platform can be developed. This platform could be used to provide access to data and functions made available by the different information sources that are effectively aggregated by the aggregated integration platform. Applications from all levels of the automation pyramid can retrieve use-case specific information from the information sources via the aggregated integration platform, and use this information to perform tasks along the life-cycle of the automation system. Therefore, time and resource consuming couplings with the information sources can be minimized. Such an aggregated platform could also ensure that if standardised information models are being aggregated, the overlaps between them are solved resulting in a single standardised aggregated information model. Therefore the consumer applications would only need to be aware of a single standard information model (that of the aggregated information model), thereby eliminating the need to know all the standards on which the information exposed by the information sources are based on.
Therefore the goal of this thesis is to find a generic approach that can be used to solve the overlaps (if any) that exist between the information models that are currently in use in the industry. This approach should entail a mechanism to aggregate several information models into an aggregated information model which should then be exposed by an aggregated integration platform.

1.3 Structure of the thesis

The structure of this thesis has been depicted with the help of a UML Activity Diagram ([Run19]) in Figure 1.2. The individual chapters of this thesis are represented as *Activities* (rectangles with rounded corners) that were carried out during the course of this thesis. The deliverables of some chapters are represented as *Objects* (rectangles).

Chapter 1 provides a short introduction to this thesis and explains the manner in which this thesis has been structured. Chapter 2 discusses the management of cyber-physical production system which serves as the basic background on which this thesis is based. It starts with an introduction to complex automation systems, their types structure and the life cycle of such systems. Then it outlines what cyber-physical production systems are and what their structure looks like. Thereafter, Chapter 2 outlines the purpose of information modeling in a cyber-physical production system and the related activities in during plant life cycle. The last section discusses how information is integrated in such a system and the resulting challenges of the same. Finally a possible solution concept is introduced and the task for this thesis is set.
Chapter 3 explores the state of the art with respect to the methods and tools that are currently used for information integration. It takes a deeper look at the

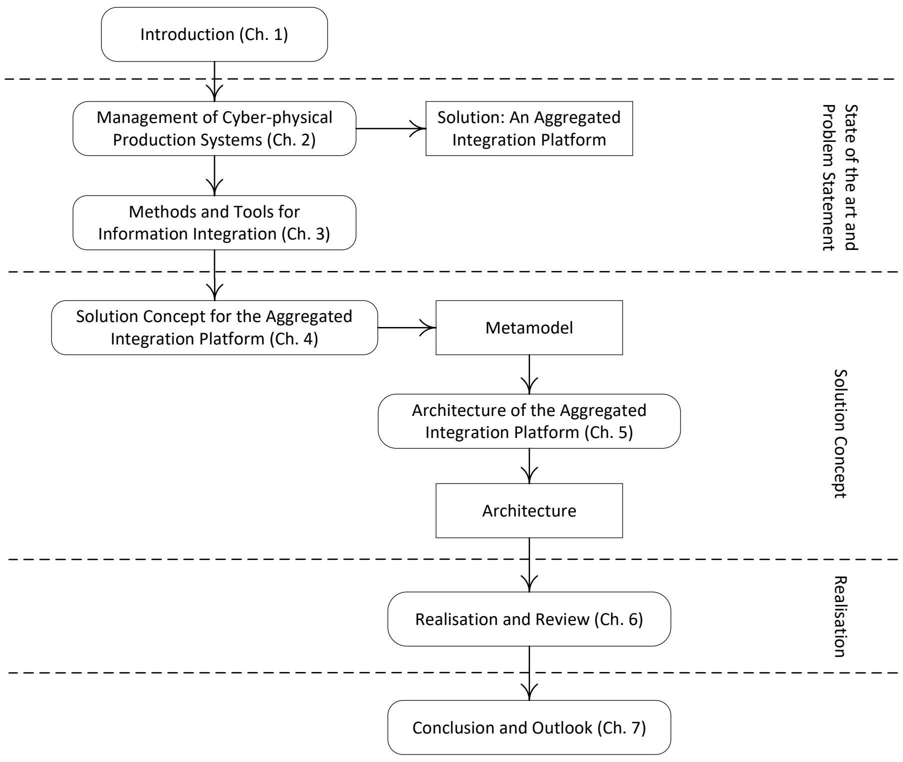

Introduction (Ch. 1)
Management of Cyber-physical Production Systems (Ch. 2)
Solution: An Aggregated Integration Platform
Methods and Tools for Information Integration (Ch. 3)
State of the art and Problem Statement
Solution Concept for the Aggregated Integration Platform (Ch. 4)
Metamodel
Architecture of the Aggregated Integration Platform (Ch. 5)
Architecture
Solution Concept
Realisation and Review (Ch. 6)
Realisation
Conclusion and Outlook (Ch. 7)

Figure 1.2: Structure of the Thesis

information modeling process, the methodologies used and the steps that are usually followed. Chapter 3 then analyses efforts that have been carried out in the direction of providing a common landscape of information models in the automation scenario and discusses some of the approaches with an eye to use one of the approaches for this thesis.
Chapter 4 discusses the solution concept for the aggregated integration platform. After presenting the basic concept, an analysis of the actors and use-cases is carried out. Based on this analysis, the requirements for the aggregated integration platform at set. To understand the requirements better, an example aggregation scenario from the process industry is discussed with emphasis on the merging of information models. This merged information model was demonstrated with graphs information models created for the example. In conclusion, this chapter delivers a meta model of information aggregation.
Based on the aforementioned meta model, the architecture of the aggregated integration platform is developed in Chapter 5. This architecture defines the structural

and behavioral model of the platform as well as the interfaces needed between the different modules of the architecture.
Chapter 6 details the prototype developed based on the architecture from Chapter 5. It explains all the modules that together build up the prototype and the interactions between them. In conclusion, the developed prototype is evaluated against the requirements set in Chapter 4. Chapter 7 summarizes the work carried out within the scope of this thesis and provides an outlook to the future research work that can be carried out on the basis of the outcome of this thesis.

Chapter 2

Management of Cyber-physical Production Systems

Contents

2.1 Complex Automation Systems 7
2.1.1 Types of Automation Systems 8
2.1.2 Structure of Automation Systems 9
2.1.3 Life Cycle of Automation Systems 11
2.2 Cyber Physical Production Systems 14
2.2.1 Structure of a CPPS 15
2.2.2 Information Modeling in a CPPS 16
2.3 Information Integration in a CPPS 17
2.3.1 The resulting connection mesh 18
2.3.2 Enterprise Application Integration 19
2.3.3 Aggregated Information Model 21
2.4 Summary and Task Definition 24

2.1 Complex Automation Systems

In today's production scenario, automation plays one of the most important roles. Germany has taken a leading role in the establishment of the 'Industrie 4.0' concept, which focuses on Smart Industries. With the increase in the use of smart sensors, actuators and measuring devices, the automation systems, although becoming much more capable, are getting more complex with time too.

The following sections will be outlining the basic concepts of complex automation systems, the structure of such automation systems and further introduce the idea of cyber physical production systems.

2.1.1 Types of Automation Systems

In order to understand the different types of Automation Systems, a clear understanding of Automation Technology is needed. The term ‘Automation’ was essentially formed from two greek words namely ”Auto” (self) and ”Matos” (moving). This means that in essence it is a system of self-moving parts. However automation systems are much more than the traditional meaning of the word. Automation systems perform significantly better than their manual counterparts in terms of power, precision and speed of operation [Kha09].

Definition 2.1 *Automation is a set of technologies that results in operation of machines and systems without significant human intervention and achieves performance superior to manual operation [Kha09]*

Automation Systems can be divided into different categories based on the level of integration and flexibility in the manufacturing process operations. Some definitions of the different types of Automation Systems are outlined as follows:-

Fixed Automation : It is the type of Automation used in production systems where the volume of production is high and it needs dedicated equipment designed to be efficient for a fixed set of operations. Used in Continuous flow and Mass Productions systems e.g. Distillation Process, Paint Shops, Transfer lines etc[Kha09].

Programmable Automation : Type of Automation where the sequence of operations and the configuration of the machines could change with the help of electronic controls. This might need non-trivial effort in reprogamming of the machines. Used in Batch processes with medium to high product volume e.g. Steel Rolling Mills, Paper Mills etc [Kha09].

Flexible Automation : Type of Automation used in Flexible Manufacturing Systems (FMS) where operations are mostly controlled by computer algorithms. These computers automatically convert the high level commands given to them by human operators, into low level changes which are then communicated to the production machines. The processing instructions are then carried out after automatically loading/unloading the required tools for the specific operation. The products are then transferred to the next machine automatically. Such systems make use of Computer Numeric Control (CNC) machines and Automated Guided Vehicles (AGV) etc [Kha09].

Integrated Automation : It represents complete automation of a production system, with each process being controlled by computer processes and communicating with other processes in a streamlined and effective manner, thereby improving the efficiency of such a system manifold. It is a complete system which may integrate business processes along with manufacturing processes and their monitoring and diagnostics too. This is achieved with the help of information and communication technologies. It involves components of varied complexities working together as a single unit. Examples of such systems are found in Advanced Process Automation Systems and Computer Integrated Manufacturing (CIM) [Kha09].

2.1.2 Structure of Automation Systems

In an industrial process, there are several components responsible for various functions related to Instrumentation, Commissioning, Control, Supervision and Operations Management. All of these inter-communicating components make up an Industrial Automation System. These components interact with each other to make the entire process more streamlined and efficient functioning of the process as a whole. Typically an Automation System can be represented by the 'Automation Pyramid' shown in the figure below (Figure 2.1). The Automation Pyramid is represented with six levels namely, *the Production Process, the Field Level, the Control Level, the Process Control Level,the Operations Control Level, the Enterprise Level* [Rot16], which are explained in a bottom up manner as follows:

- *The Process Level* : The bottom most level is taken up by the production process that is being automated.
- *The Field Level* : The Field Level consists of the sensors and the actuators that interact with the Production Process directly. The sensors sense trigger signals and communicate the same to the Control Level [Rot16]. The actuators receive instructions from the Control level and then perform the respective actions on the Process level. The communication between the field level and the control level takes place over the communication system of the field level e.g. Fieldbus protocols like Profibus, HART etc. The time window for the systems at this level are in the order of seconds and smaller [Com+13].
- *The Control Level* : The Control Level defines the logic which converts the input signals received from the sensors (in the Field level) to the outputs

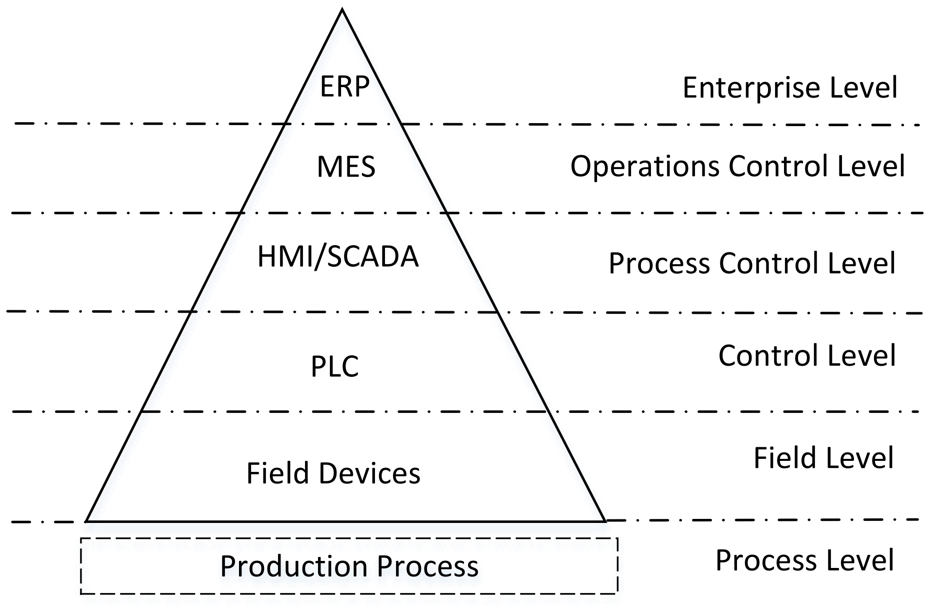


Figure 2.1: The Automation Pyramid - Hierarchy of Automation Systems (adapted from [Rot16])

which are relayed to the actuators (in the field level) [Rot16]. This level consists of control components such as Programmable Logic Controllers (PLCs) which run the logic programs and produce the output signals for the actuators. The time frames being considered by the systems at this level are in milliseconds, seconds, minutes and hours [Com+13].

- *The Process Control Level* : The components of the Process Control Level are applications for Process visualization and management, for the configuration and programming of the process control systems [Rot16]. Therefore, at this level an interaction between human beings with controlling and monitoring applications takes place. Here systems such as Human Machine Interaction (HMI) and Supervisory Control and Data Acquisition (SCADA) are used.

- *The Operations Control Level* : The components of the Operations Control Level initiate and monitor the Production Process with the help of Production Orders. Typical examples of systems on this level are Production Control Systems e.g., Manufacturing Execution Systems (MESs)) [Rot16]. The time frames considered by the systems at this level are given in seconds, minutes, hours, shifts and days [Com+13].

- *The Enterprise Level* : The Enterprise/Company Level consist of business systems which monitor the production process at the highest level of abstraction. This level typically consists of Merchandise Management systems (e.g.Enterprise Resource Planning (ERP)) or logistic systems [Rot16]. The time scales being considered by the systems at this level are days, weeks and months [Com+13].

2.1.3 Life Cycle of Automation Systems

For the establishment and operation of automation systems until decommissioning, various different disciplines have to work together. In order to make this plant life cycle manageable, phase plans have been established that divide the entire life cycle into manageable parts. [NAM19] describes such a phase plan in Figure 2.2. The individual phases are to be implemented with the focus on the construction of automation systems. Accordingly, the plan starts with collection of the project requirements and scope definition, and ends with the completion of the project - the fully implemented functional system.

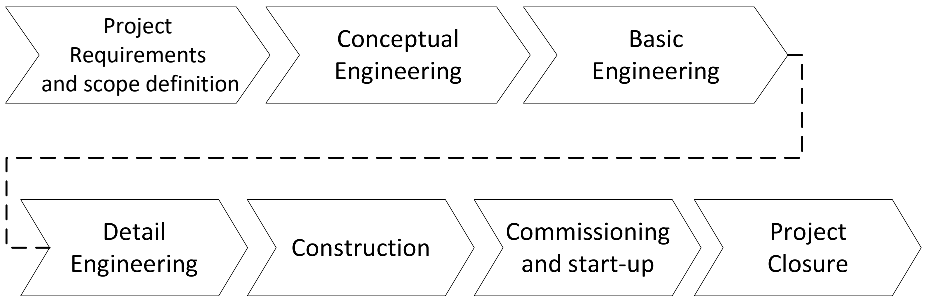


Figure 2.2: Phase plan as per [NAM19]

However, this phase plan leaves out the Operating Phase entirely, since within the scope of [NAM19], the project is considered to be concluded after the commissioning and start-up phase. From the point of view of the plant operator, the Operating Phase takes the center stage because of the value being added to the automation system in this phase. An extended phase plan which also includes the Operating Phase has been shown in Figure 2.3. The Operating Phase follows the Project Completion phase and it encompasses the *Service, Maintenance, Modification* and *Shutdown* phases, finally ending with the *Disposal* phase.

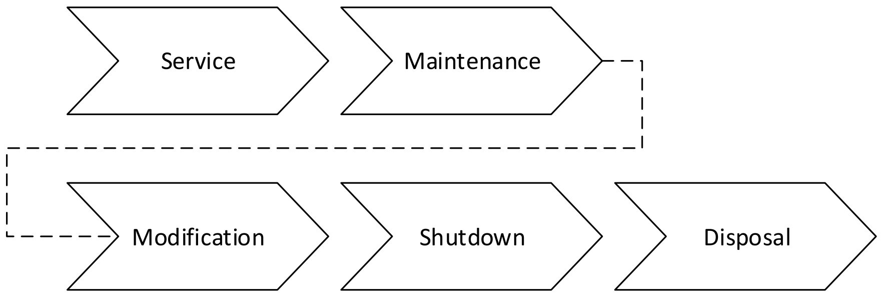


Figure 2.3: Phase plan of the Service phase as per [FMS04]

With the advent of IIoT, the life Cycle of automation systems and its components has also evolved. In [Soo+18] the authors postulate a so-called IoT device lifecycle management model. The IoT device lifecycle management model (Figure 2.4) refines the commonly used product life cycle models namely Beginning of Life, Middle of Life and End of Life, and restructures the same with respect to the components that constitute automation systems. This model uses the well known Business Process Model and Notations (BPMNs) to represent the model. From a conceptual point of view, this model encompasses the main points of the aforementioned phase plans. The following is a discussion on the different phases and how they relate to the phase plans mentioned in the previous paragraphs and other literature on the same.

Plan and Design As part of the basic assessment, the project objectives are defined as well as the approximate costs are estimated. The aim is to ensure the feasibility of the project. The subsequent preliminary planning defines the plant at a conceptual level and refines the cost estimation. The basic planning defines the automation functions as well as their technical realization so that the tendering of the plant can take place. The tasks of the execution planning are the selection of the field devices, the facilities of the control and automation system including the how the information will be integrated and the preparation of the required assembly documents. After the plan and design phase is complete, the plant can be built.

Provisioning During this phase, the configuration or programming of the software of the automation system takes place. This also includes the complete development of information aggregation platforms as per the chosen technology for the particular application scenario. After the preparation and execution of the assembly, the functional test ensures that the target of the functional system has been achieved.

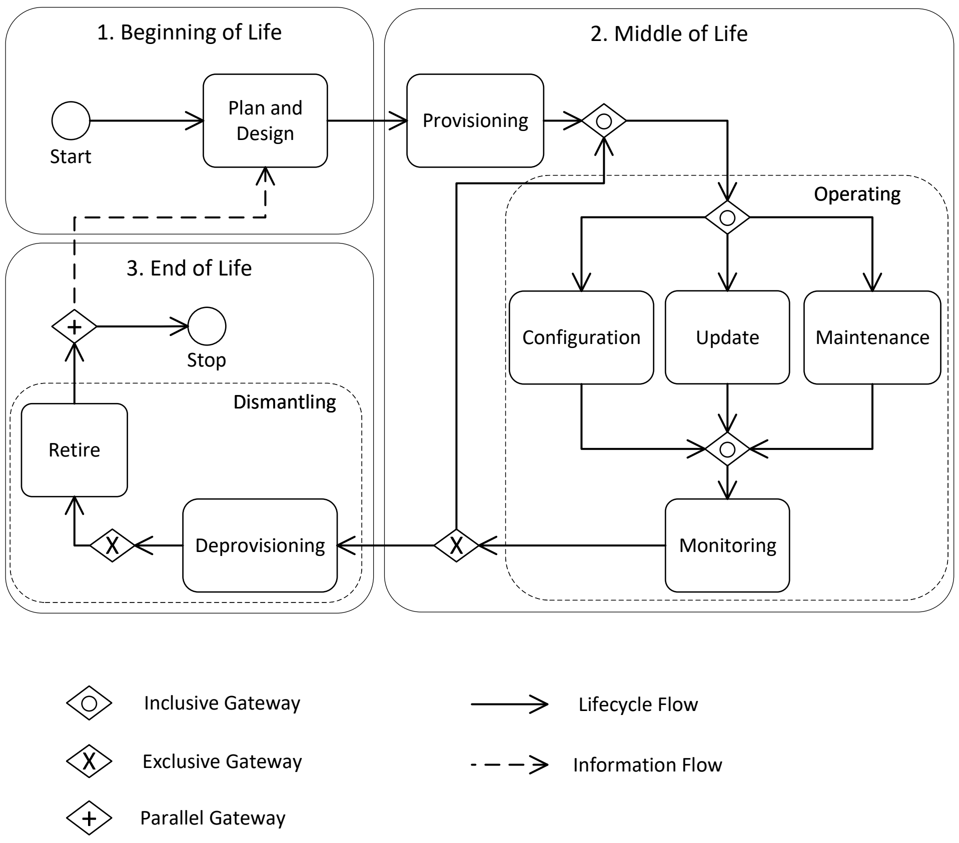


Figure 2.4: The IoT Device Lifecycle Management Model (adapted from [Soo+18])

Operating The *Configuration*, *Update*, *Maintenance* and *Monitoring* stages collectively form the *Operating* phase. The production-capable plant forms the starting point of the operating phase. During operation, the service or more specifically monitoring takes place, i.e. continuous monitoring of the processes and of the automation equipment. If spontaneous errors occur, which lead to a malfunction of the operation, a diagnosis is carried out to identify the possible causes. After subsequent repair, the system is transferred to regular operation. This type of plant operation is also referred to as Reactive Repair [Eme03]. In the case of preventive maintenance [Eme03], on the other hand, a given maintenance schedule determines the time of scheduled decommissioning in order to carry out the pending maintenance measures. Subsequently, the system is restarted. In the case of predictive, condition-based maintenance [Keg07], the decommissioning does not take place at predetermined time intervals, but on the basis of state variables which are e.g. determined by extrapolation of the date of the next scheduled decommissioning and maintenance. Likewise, after a scheduled shut down, changes or extensions of the plant are made, e.g. to adapt to the production of a new product.

Dismantling The *Deprovisioning* and *Retire* stages collectively form the *Dismantling* phase. If in case the further operation of the plant is no longer feasible - mostly due to economic considerations, the life cycle ends with the Dismantling of the plant and its components.

2.2 Cyber Physical Production Systems

Cyber Physical Systems (CPS), as the name suggests, are systems made up of *Cyber* (Virtual) as well as *Physical* (real-life) components. To elaborate, CPS are systems that function in a flexible, cooperative and interactive manner. *Cooperation* in this context entails system-system-cooperation, where the non-human components of the system are connected to each other and exchange data and information in order to continually increase the efficiency of the overall system. *Interaction* refers to the human-system-cooperation where the human components of the system interact with the other components of the system e.g. with the help of a *Human Machine Interface* (HMI) [Mik14] . The underlying key enabling technology which acts as the backbone for the CPS is the *Internet of Things* (IoT). The term IoT was first used in 1999 in MIT Auto-ID Labs [WW18]. In simple terms, IoT can be defined as a network of interconnected uniquely-identifiable components (*things*) of a system which can communicate with each other based on standard communication protocols. In a broader sense, the IoT architecture can be visualized to consist of four layers: the sensing layer, networking layer, middleware layer, and application layer. The *Sensing Layer*, which contains the sensors responsible for sensing and capturing real-time information. The *Networking layer* connects all the components together with the help of a physical network. This physical network is used to exchange data and information between the various interconnected components. The *Middleware layer* consists of hardware and software platforms with the help of which the real-time data and information is transmitted. The *Application layer* contains the algorithms which implement the system functions to enable IoT for manufacturing and industrial use-cases [WW18].

Industrial Internet of Things (IIoT), in comparison, is a more specific subdomain of IoT focusing primarily on manufacturing and production processes in industrial automation. It adds more value to these processes by globally connecting the various aspects of the system including but not limited to the machines,assembly lines and factories [Jes+16]. Also known as the *Industrial Internet*, it is a network of devices communicating with each other with the help of various protocols used for industrial communication. This network makes it possible to monitor, analyze and diagnose the system in a much more efficient way than was possible before [OT17]. Thus IIoT plays the role of the enabling technology for CPS for the manufacturing and production industries, also known as *Cyber Physical Production Systems.*

Definition 2.2 *Cyber Physical Production Systems (CPPS): A CPPS can be defined as an interconnected heterogeneous and distributed system of devices and components which seamlessly communicate with each other to exchange data and information in a production scenario, in order to continuously monitor the physical production processes, perform diagnostics and make the entire production process more effective and efficient [Mon14].*

2.2.1 Structure of a CPPS

The structure of a CPPS can be depicted as the below Figure (2.5). To discuss the flow of the information between the different components of the CPPS, the data and information about the actual Production Process is being collected by the *Sensor Elements* of the system. This is then processed by the *Data Processing and Transmission* unit which determines the action to be taken by applying the business logic to the input data. It then provides the respective output to the *Actuating Elements*, which then carry out their appropriate tasks as outlined by the Data Processing Unit. These three units, namely the Sensor Elements, Actuating Elements along with the Data and Transmission Unit form the core of the CPPS. The system functions seamlessly with the support of additional cooperative units such as the *Local Services* unit which handles the cooperation of the core system with the services in the local environment, and the *Human-System Cooperation* unit, which handles the interaction with the human components of the system. In addition to these external units, the core is also connected to an *Internet of Things / Internet of Services* which plays the role of further analyzing the data thus generated in order to formulate a behavioral description of the production process. This data and information analysis is again made possible with the help of the *System-System cooperation* and the *Global Services* unit, which in this case could be a global repository of services which encapsulates the shared collective intelligence of the knowledge being gained by similar production systems, and can be used to make the process more efficient with time.

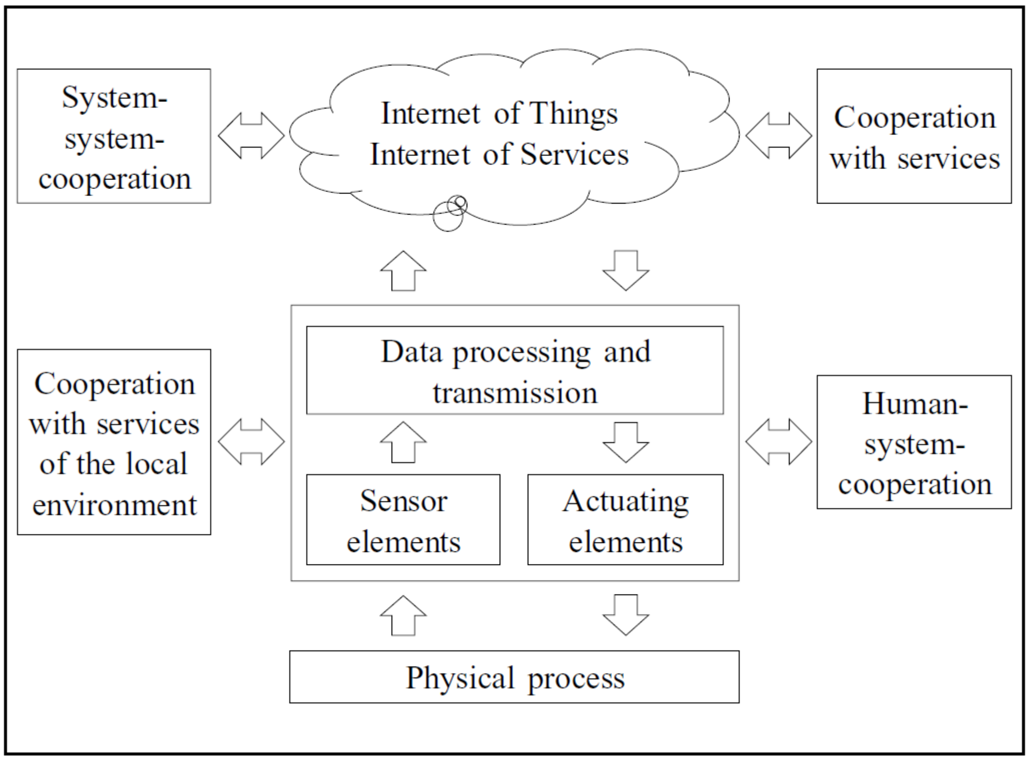

System-system-cooperation
Internet of Things
Internet of Services
Cooperation with services
Cooperation with services of the local environment
Data processing and transmission
Human-system-cooperation
Sensor elements
Actuating elements
Physical process

Figure 2.5: Structure of Cyber Physical Production Systems [Mik14]

2.2.2 Information Modeling in a CPPS

The *Cyber* in CPPS is largely dependent on how the components of the production system have been modeled in the *virtual* domain. Information modeling is one of the most important aspects of an automation system. It determines how the entire available information about the system as a whole could be structured and represented in an accessible, understandable and efficient manner. Within the context of this thesis, this term has been defined as below, while referring to [MLD09], [RQ+12]:

Definition 2.3 *Information Model : The Information Model of a CPPS is a representation of the entities of the system, their data and behavioral semantics, and the relationships, if any, between the various entities that make up the system. It usually represents a system with a particular functional domain in mind.*

As the definition above suggests, not all of the information about a system is modeled in a single information model. Before the modeling is carried out, the *purpose* of the information model is defined, which means that the a certain aspect of the CPPS is being concentrated upon, and only that particular aspect is represented in the information model. For example, let us consider the scenario of a automotive manufacturing plant. An information model of this plant can be created which describes the physical structure of the plant i.e. the physical location of the different components (e.g sensors, actuators, conveyor belts, PLCs etc.) of the plant with respect to each other and their topographical structure. This model would define the topological hierarchies between the different components and their physical location in the plant. Another information model could concentrate on the real-time data being exchanged between the devices of the same plant. Therefore, although the two information models model the same physical plant, their modeling purposes could be different. Each information model would be conceived with the aim to show a particular aspect of the system. When an information model is ready, it is exposed with the help of an *Integration Platform*. Within the context of this thesis, this term has been defined as below:

Definition 2.4 *Integration Platform : The software component that implements and exposes the information model that it is designed for. This is usually achieved with the help of Software Development Kits (SDKs) which are designed specifically for the purpose of information integration and these SDKs provide the tools needed in order to communicate data and information to and from the Integration Platform.*

2.3 Information Integration in a CPPS

Information integration plays an important role in how data and information is handled in an automation system. Now, this information integration can only be helpful if it is efficient, consistent, and easily accessible by the applications that need this information and process them further in order to achieve a specific goal. In order for the information model to be consistent, information modeling standards were created. Information models can be conceived based on these modeling standards. Therefore, based on the purpose of modeling, the modeler can base his model on the appropriate modeling standard and can build the integration platform in a standardized manner so that the access to the information exposed by the model (by respective applications) can also be consistent and standardized. In today's day, the information exchange between the different layers of the traditional automation pyramid is gradually changing. As shown in Figure 2.6, in a CPPS, the information sources from the different levels are connected to each other and there is a possibility to information exchange between them and the sophisticated applications which can gather and use the information from these sources, all at the same time.

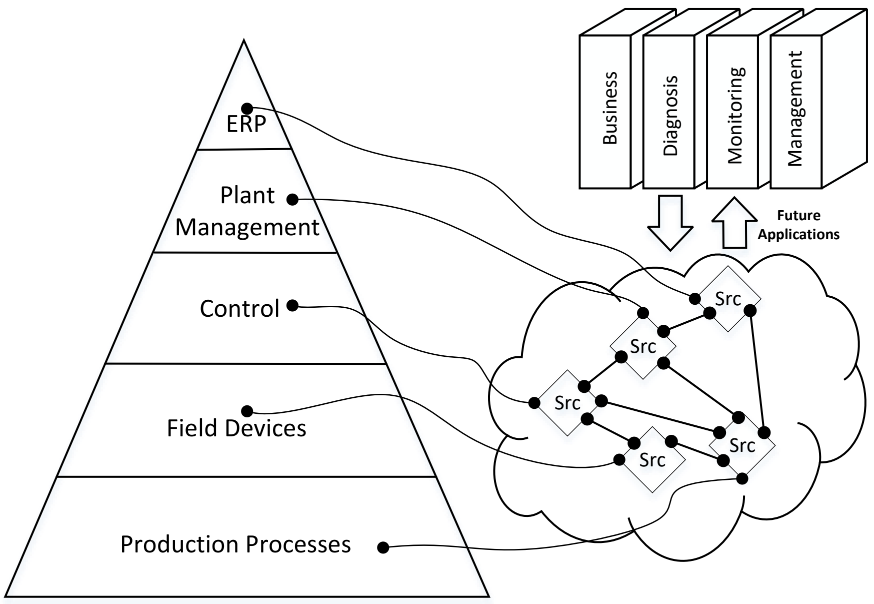


Figure 2.6: Automation Pyramid in a CPPS perspective
[Col+14]

2.3.1 The resulting connection mesh

With the advent of Industrie 4.0, the different components of the CPPS are getting smarter and smarter. The term 'smart' here is to suggest that the with the massive increase in processing power and reduction of size of the components, the individual components of the automation system (e.g the sensors or actuators at the field level or the control systems controlling them etc.) are now much more capable than before. As a consequence, components and subsystems of the automation system are already equipped with their own information models, thus providing data and functionality. There is an obvious advantage to this trend, which is that such components can already be accessed directly even before commissioning of the entire automation system is completed. This further drives the collaborative effort of the highly distributed system components. However, this would then result in a system with (fully) meshed connections between functionality providers and the respective consumers in a CPPS (Figure 2.7).

Automation components and subsystems with an inbuilt information model does have its own advantages but at the same time there are some disadvantages too. For example, throughout the life cycle of an automation system, different applications (or tools) need the access to the data and information provided by

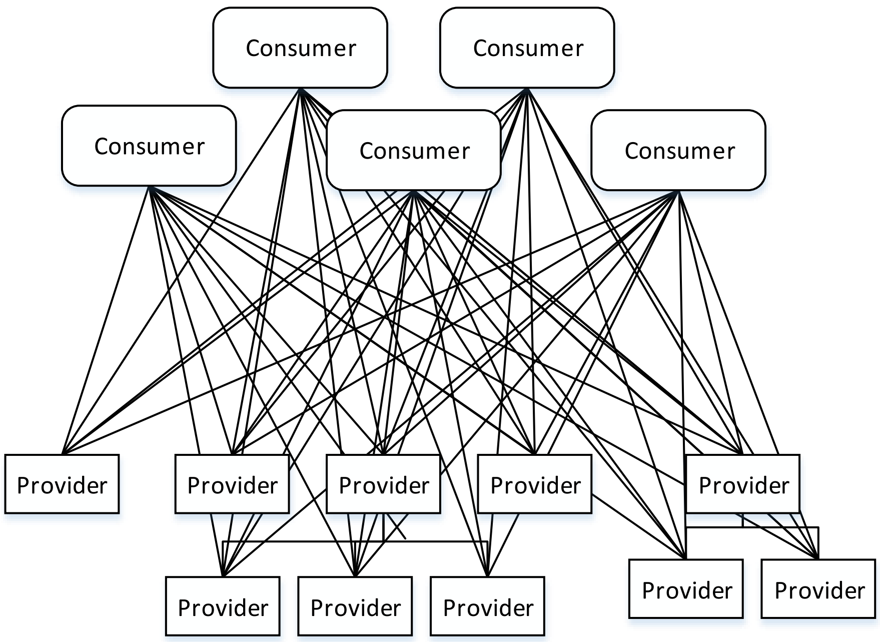


Figure 2.7: Resulting Communication Mesh

these components and subsystems. This problem was solved quite practically and efficiently in the scenario of a single centralized integration platform as the consumer applications would just need to connect to the central platform and get access to the data and information that the platform integrates from underlying components or subsystems. However with the advent of the distributed and decentralized information sources as in the case of a CPPS, the consumer applications need to connect to each individual information source and access the information relevant to it. Moreover, from a security standpoint, each of these connections need to be managed individually as per the security guidelines and requirements.

2.3.2 Enterprise Application Integration

The domain of business information systems has managed to develop a solution to solve a similar problem of coupling variety resulting in meshed networks. There is a similar challenge of heterogeneous environments from which data and information need to be accessed by multiple applications for further specific processing. As shown in Figure 2.8a, this situation results in a fully meshed network and a large number of connections between the individual systems that act as the provider and consumers of information.

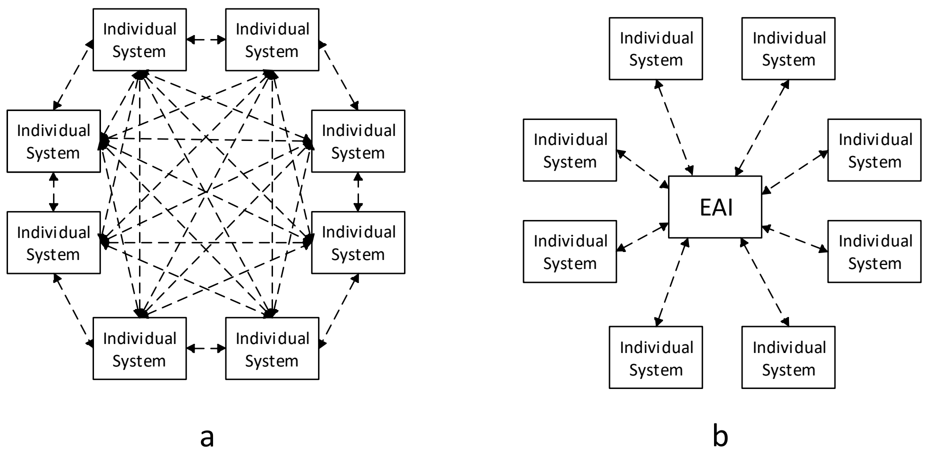


Figure 2.8: Enterprise Application Integration

Equation 2.1 shows the number of connections in such a fully meshed system. As is clear from the outlined equation, the number of connections is a quadratic function of the number of individual systems.

$$N = \frac{n(n-1)}{2} \subseteq O(n^2) \tag{2.1}$$

To reduce the number of connections in such a scenario, a concept of *Enterprise Application Integration (EAI)* was established [Aie06]. The basis of this concept is a intermediary integration platform that assimilates the information from all the other available enterprise applications. All the requests and responses of data and information are through this intermediate layer. The intermediate layer receives the request and then forwards it to the appropriate information provider. In the same way when it receives the response corresponding to a particular request, it then forwards the response to the consumer application that had requested for it in the first place. The introduction of this intermediate layer reduces the number of connections to the same as the number of communicating components (Figure 2.8b) . The number of connections then becomes a linear function of the number of communicating components of the system (Equation 2.2).

$$N = n \subseteq O(n) \tag{2.2}$$

2.3.3 Aggregated Information Model

In order to solve the resulting connection mesh due the ever increasing capabilities of automation components (as discussed in Section 2.3.1), there is a need for an architecture similar to the EAI (Section 2.3.2), which provides for an intermediate Aggregation layer which aggregates other information models (which may be exposed directly from the underlying components or subsystems). This aggregation layer will then provide an "illusion" of a centralized integration platform. This will then reduce the previously discussed connection complexity.
At its core, the aggregation layer will work in a way similar to the EAI scenario solution i.e. it will receive requests from consumer applications and forward them to the appropriate underlying integration platform. After the request is processed by the underlying integration platform, it will be given back to the aggregation layer. The aggregation layer will then forward the response to the actual consumer. This then provides an added advantage that the consumer applications should no longer need to "know" about all the relevant underlying integration platforms. Furthermore, the security aspect can be centralized at the aggregation layer and can be handled in a much more efficient manner. This concept will be henceforth termed as the Aggregated Information Model. Thus, the following definitions have been formulated within the context of this thesis.

Definition 2.5 *Aggregated Integration Platform : An Aggregated Integration Platform aggregates the information models exposed by other integration platforms. It acts as a layer of abstraction between the actual sources of information (integration platforms being aggregated) and the consumers of information (applications that would otherwise need to connect to the individual information in order to get the information). The entire communication between the information sources and sinks is via the aggregated platform.*

Definition 2.6 *Aggregated Information Model : The information model exposed by the aggregated integration platform which is constructed by assimilating the information models of the integration platforms that are being aggregated into a single information model. This means that the aggregated information model contains all the type, structure and relationship information about the components of the information models that are being aggregated in a single consolidated model.*

As can be observed from the Figure 2.9, the Aggregated Information Model solves the problem of connection complexities by acting as the gateway between the consumer applications and the underlying information providers.

The number of required connections in such a case will be equal to the sum of the number of underlying information models being aggregated n_{um} and the number of applications that consume the information n_{ca} (Equation 2.3).

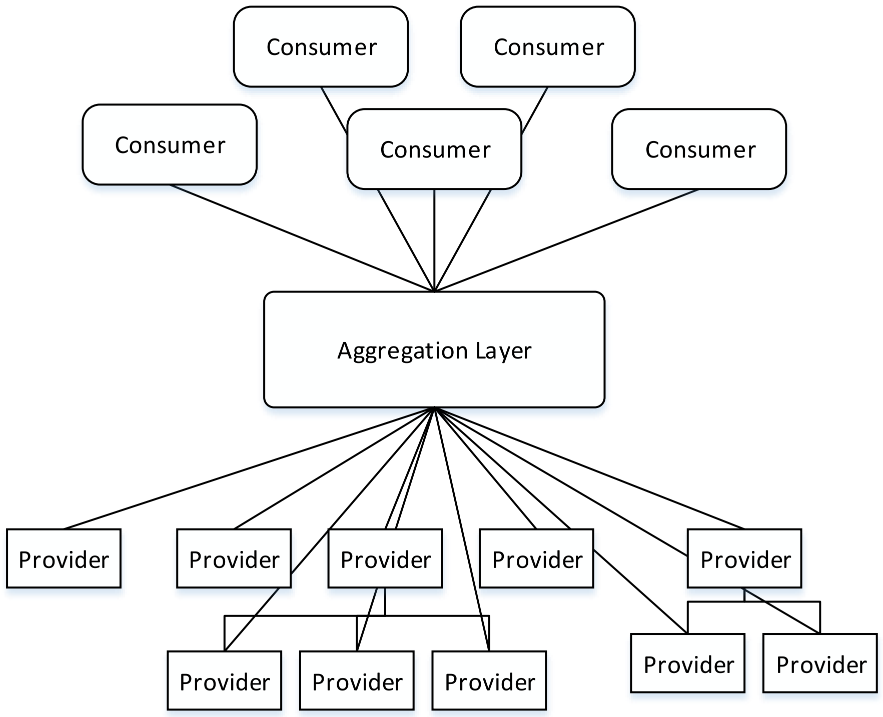


Figure 2.9: The Aggregation Layer

$$N = n_{um} + n_{ca} \subseteq O(n) \tag{2.3}$$

There are a number of requirements for the creation of an Aggregated Information Model. A detailed explanation of the requirements will follow in Chapter 4 of this thesis. While most of these requirements have been adapted for the Aggregated Information Models based on the generic requirements for information modelling as per [BPV12], [Mat+20], [GH20], [ATE19], some have been derived from standard software development best practices [SVC06]. The most important requirements have been outlined below:

Domain Independence The Aggregated Information Model should ideally be usable in the different domains of the automation industry i.e. the concept should be open enough to include process industries as well as manufacturing and production industries. Specific requirements pertaining to any particular domain should not be included.

Completeness This requirement states that when an information model is being aggregated, it should be done in its entirety. The entire information which could be represented by the stand-alone information model, should also be accessible through the Aggregated Information Model. Any additional functionalities provided by the underlying information providers should also be provided by the aggregated integration platform.

Non-Redundancy The Aggregated Information Model should not contain any elements that are redundant. This is to say that when other information models are being aggregated, it could be possible that the same entity exists in two different information models. The information modeled for the said entity would still belong to different domains. However, when the virtual representation of such an entity (which exists in two different information models) is aggregated into a single aggregated information model, there should be no duplication i.e. the same entity should not have two different virtual representations. The aggregated information model should contain a single virtual representation of the said entity and the information about the same should be aggregated and clubbed within the same virtual representation.

Openness The Aggregated Information Model must be open. This means that on the one hand, it should be independent of the organizations providing the underlying information model and on the other hand End-User applications can openly access the Aggregated Information Model.

Transparency Access to the data and functionalities of the underlying information models should be transparent. This means that the end-user should "see-through" the underlying mechanism of how the aggregation layer is handling the information requests and responses and a hassle-free communication path is established. The consumer application should be able to access the information from the aggregated model in the same way as though he would have accessed it directly from the underlying information model.

Platform Independence The different systems in this particular scenario i.e. the integration platforms to be aggregated as well as the applications which need access to the information, are all heterogeneous i.e. they might be running on different kinds of hardware and software platforms. Because of this heterogeneous nature of the system, any specific Operating System or Hardware configuration cannot be fixed which should be used for all the components of the system. Therefore the concept of the Aggregated Information model should also be platform independent so that it can be implemented on all possible hardware and software platforms.

Semantics The aggregated integration platform should provide the possibility of semantics based operation. This is to say that the aggregated model should be able to be used by specific applications, without the need for prior configuration. It should be flexible enough to aggregate other information models without affecting the normal functioning of the system at any point in the operating phase of the automation system.

2.4 Summary and Task Definition

In order to structure the life cycle of automation systems, there exist Phase models which consist of several phases that the automation system goes through in succession. One such general life cycle phase model which outlined the different stages of an automation system has been put forth by NAMUR NA 35 specification. In light of Industry 4.0, other life cycle models have been proposed. These life cycle models represent all the important phases IIoT devices go through right from its Planning up until its Dismantling phase.
In light of the concepts of Industrie 4.0 and Cyber Physical Production Systems, information integration plays a major role. The way the data and information of and about the components of an automation system are represented is determined by the information model of the system. This information model is then exposed by the integration platform which provides access to the data and functionalities provided by the components of the automation system. Applications then connect to the integration platform and consume the information for further processing for maintenance and diagnostic purposes as outlined by the requirements of the application scenario.
With the steady increase in the processing power and capabilities of devices and automation components, there is a growing trend of distributed integration platforms, as the components have their own integration platforms. However the applications that need access to the information goes on increasing. This gives rise to a complex fully meshed communication network. In this case if an application needs access to multiple such ”smart” components, it needs to access the information models of each of these components separately in order to assimilate the required information.
To solve this problem of the fully meshed connections, an Aggregated Information Model could be the appropriate solution. The Aggregated Information Model will comprise of the aggregation of all other information models being exposed by the distributed integration platforms. Thereby it will provide an illusion of a central integration platform and will provide the data and functionalities of all the inte-

gration platforms it aggregates, all in one place.
Accordingly the goal of this particular thesis is to design and develop a concept and an architecture for such an Aggregated Integration Platform which could successfully aggregate other information models.

Chapter 3

Methods and Tools for Information Integration

Contents

3.1 Development of Information Integration 28

3.1.1 The General Information Integration Process 28

3.2 The Information Modeling Process 32

3.2.1 Conceptualization of Information Models 33

3.2.2 Modeling Methodologies 35

3.2.3 Modeling Steps 37

3.3 A Common Landscape of Information Models in Automation 40

3.3.1 Reference Models 41

3.3.2 Model Universals 41

3.3.3 Concept for consistent use of Engineering Models in Automation 42

3.3.4 SemAnz4.0 42

3.3.5 RAMI 4.0 44

3.4 OPC UA - An Enabler for Industry 4.0 46

3.4.1 Motivation and Overview 46

3.4.2 OPC UA Base Layer 47

3.4.3 OPC UA Model Extensions 51

3.5 Summary 53

3.1 Development of Information Integration

The development of the integration platform is directly related to the different information sources in an automation system. As entailed in Section 2.1.2, there are different levels in an automation system. The information sources and the way to access that information depends upon the level where that particular information source sits.

At Level 1 - the Field Device Level : When industrial automation had just started, the field devices were fairly simple components which worked on analog signals e.g. the sensors would simply transmit analog signals to the control systems. The field devices were being manually configured at the site. After this phase, came the field devices which had a screen and a microcontroller to monitor and configure the same. Today, we have intelligent field devices which have an embedded operating system. These intelligent field devices have the information model exposing their data and information in a structured manner.

At Level 2 - the Control Level : Over the course of time, the control systems have also improved considerably. Before the 1950's most control systems were analog and were a simple combination of "on-off" arrangements of analog-switches and relays. Then came the age of Numerical Control(NC) using punctured tapes followed by Computerized Numerical Control (CNC) [HAC14]. Similar to the advancements in intelligent field devices, in today's day we have highly sophisticated Programmable Logic Controllers (PLC) which not only simply control the sensors and the actuators, but are now capable of implementing complex learning based algorithms to make the whole process more efficient with time.

At Level 3 and 4 - the Plant Management and Enterprise Level : At these levels of the automation pyramid, with the growth of industrial application and technology, manual maintenance of records, orders and follow-up of the same, was replaced with use-specific computerized applications. These applications are full-fledged business solutions for a particular business scope and domain in mind. Sophisticated ERP software has now replaced many planning related tasks that an organization has to deal with. At the operational level, advanced software components have been developed that analyze the data and find patterns in the them in order to fine-tune the process even further.

3.1.1 The General Information Integration Process

The general information integration process begins with the definition of the use-case and *what* the purpose of the information integration is. The next step is to determine *how* the particular information can be collected from the different components of the system. Then the integration platform is developed and the integrated information is then exposed (visualized) as per a particular schema.

This can then be consumed by other applications depending upon the use-case. The different steps of this entire process can be visualized with the help of a V-Model derived from [VDI21] outlined in Figure 3.1.

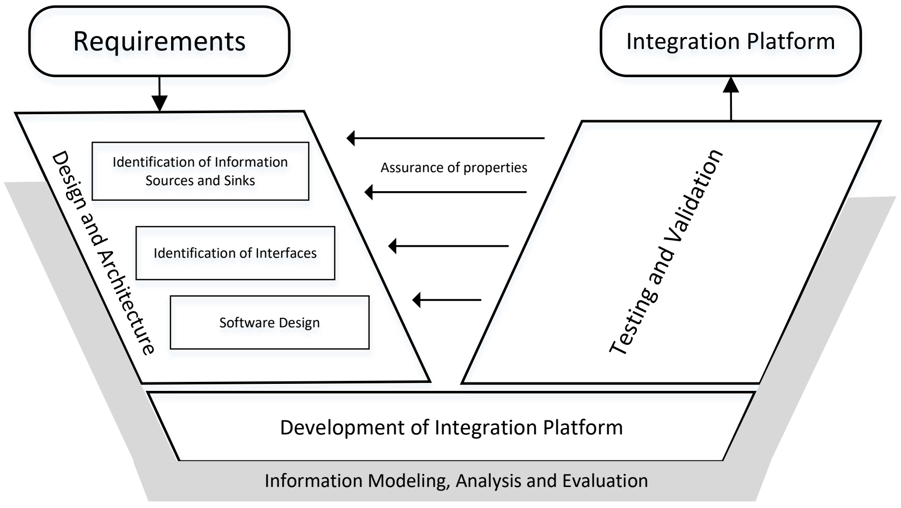

Requirements
Integration Platform
Design and Architecture
Identification of Information Sources and Sinks
Identification of Interfaces
Software Design
Assurance of properties
Testing and Validation
Development of Integration Platform
Information Modeling, Analysis and Evaluation

Figure 3.1: The Information Integration Process (adapted from [VDI21])

Requirements: In an intelligent automation system, there is a lot of data and information being exchanged between the different components at different data rates. Therefore it is important to identify the data and information relevant for the particular scenario. For example if the energy efficiency of a particular system needs to be analyzed, only the data related to the energy requirements and outputs of the automation components should be looked at. All such requirements are first collected depending upon the use-cases to be addressed and are fed into the next stage.

Design and Architecture: The next step in the V-Model is the Design and Architecture step where the system architecture for the integration platform is designed as per the requirements from the previous step. This includes the following main activities:-

- **Identification of Information Sources and Sinks**
  In an automation system, there are several sources that are generating the information and several sinks that needs to consume information in order to deliver the results.In the design phase of the information architecture, a

clear identification of the sources and sinks of/for the information needs to be identified.

- **Identification of Interfaces**
  In an automation system, different communication protocols are used between the components for information interchange. Thus, in order to accumulate information from various sources in the automation system, the correct communication interfaces need to be identified. In the case where no standard communication interfaces exist, they need to be developed as per the protocols being followed on the two sides of the communication channel. This can be explained with the help of Figure 3.2. Here the integration platform is communicating with the underlying information sources S1, S2, S3 using different communication interfaces C1, C2, C3 respectively and communicates (the information thus accumulated) with the consumer applications A1, A2, A3 with the help of communication interfaces C4, C5, C6 respectively. All of these interfaces could represent different industrial communication protocols. The translation mechanism between the respective protocols must then be handled within the integration platform.

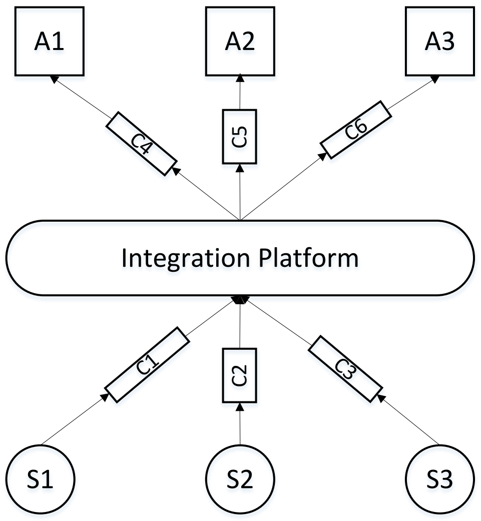


Figure 3.2: Identification of Communication Interfaces

- **Software Design**
  The different modules of the software are designed and their role in the system architecture as a whole and interaction with other module components

is defined.

Development of the Integration Platform: Finally the integration platform is developed using a suitable development technology following a particular information integration standard in mind. To sum up the process, this integration platform is the actual software component which is responsible for collecting the information from the information sources (using the communication interfaces), structuring the information in an information model, and make it available to the consumer applications (clients) for information consumption.

Testing and Validation: After the integration platform is developed, it needs to be tested validated against the requirements that had been collected at the beginning of the process. This is typically done in a modular way e.g. the individual modules are tested (Unit testing) followed by a testing of the entire system (integration testing). After passing this phase, the integration platform is ready to be used.

Information Modeling, Analysis and Evaluation The information thus compiled needs to be made available to the consumers in a structured and sensible manner. This is where the information model comes in. In order to create an information model, the virtual representations of the various entities of the system need to be conceptualized along with the relationships and semantics between these entities. The information model should also provide knowledge about the functionality carried out by these entities. As can be observed from the figure 3.1, the information modeling aspect closely follows the V-Model i.e. it has its own part to play during all the phases. This is explained further in Section 3.2.3. Figure 3.3 highlights the main components of an information model. As the V-Model enables iterative development of the integration platform, the same is also valid for the information model i.e. during the validation and assurance of properties, parts of the design and architecture can be revisited until the model is validated against the requirements.

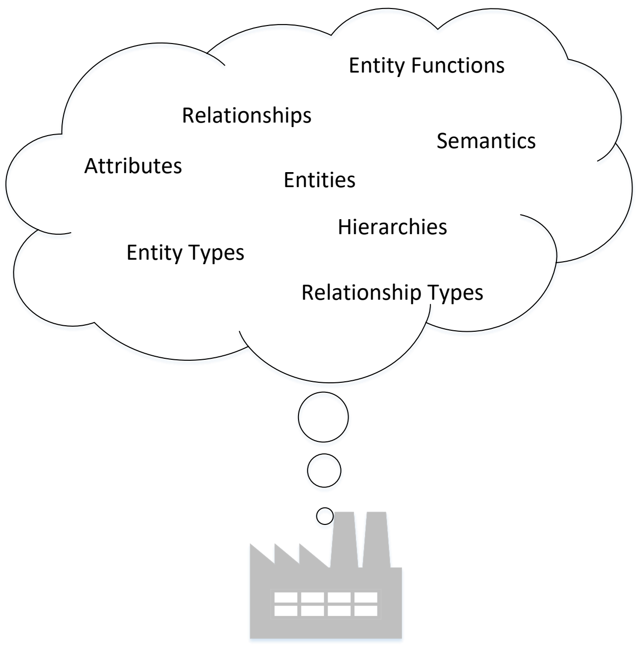


Figure 3.3: Main components of an information model

3.2 The Information Modeling Process

In the previous section, the general information integration process was discussed. The most crucial step in the whole process is understandably the information modeling step. In this section, this crucial step will be discussed in detail, keeping in mind the tools that the state-of-the-art modeling techniques provide us, which could be used to find a solution to the problem that this thesis is aimed at solving (Section 2.4).
Before diving into the information modeling process, the term "Information" needs to be defined in the context of this thesis.

Definition 3.1 ***Information***
Information is defined as the combination of Data and its semantics. Semantics encompasses the true meaning of the data and how it should be interpreted. In its raw form Data is a sequence of characters, which in essence is not useful unless its semantics are considered. The package of Data + Semantics is therefore termed as Information and can be represented with the following simple equation (3.1).

$$Information = Data + Semantics \tag{3.1}$$

3.2.1 Conceptualization of Information Models

An environment for the building of information systems was conceived in the DAIDA project [Jar+92] . It classified this information by means of four ”worlds”, as shown in 3.4. As the name suggests, the *subject world* consists of those components of the information system which are the primary subject of the modeling purpose. The *system world* consists of the information system itself and describes the details about the different implementation layers involved in it. These layers could be the functional requirement specifications, its conceptual design up until implementation details. The *usage world* consists of the boundaries and the environment within which the system should function. This entails the users and their interfaces, the tasks, activities and projects etc. The *development world* contains the processes needed to create the whole information system. It is the development team including the programmers, analysts, architects, designers, the methodologies adopted by them, the design and implementation decisions taken by them and the reasoning behind those decisions. Not only is all of this information required during the conception of the system but throughout its life cycle. Therefore the primary task of information modeling is to provide a framework to represent all the information present in the above four worlds.
In other words, the information needs to be structured in the form of computer understandable symbols which are used to model a part of the real world. [Myl98] calls these structures as *information bases* and the part of the real world being modeled as *application.* The information base may evolve over a period of time, it can keep changing to correctly model the *application.* The information base needs to be organized according to the subject matter and its contents. There it can be said that the information base can be constructed with the help of models of one or more of the four worlds shown in 3.4.

Over the years, the state-of-the-art of information modeling have moved from computer-understandable representations towards human-understandable models which can express and represent more complex application modeling purposes. Historically, this shift can be observed by the classification of the information models into three broad categories as follows:

Physical Information Models : In physical information models, the application was modeled with the help of computer programming constructs and data structures like arrays, lists, strings, trees etc. The primary disadvantage of physical information models is that the modeler has to make a choice between computational efficiency and the overall quality of the model. For example the need to choose a particular data structure and data type to represent components of the application is driven by the efficiency factor and has little to do with the real world application.

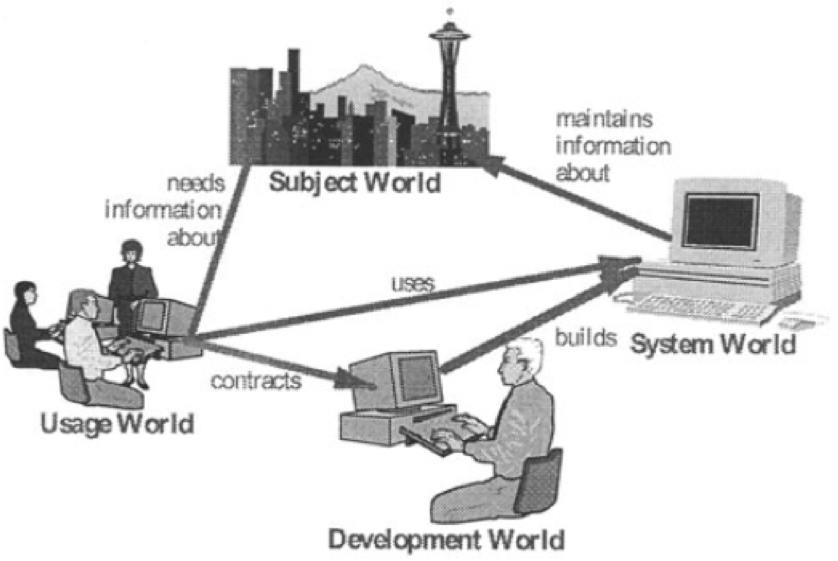


Figure 3.4: The Four Worlds of Information Systems Engineering (as defined by the DAIDA project [Jar+92])

Logical Information Models : In the early '70s, the concept of *logical data models* was introduced, where the information was represented in terms of abstract mathematical symbols (e.g.sets, functions, relations etc.), thus effectively hiding the implementation details. The relational models for databases can be mentioned as good examples of logical models. In these information models, the modeler doesn't need to consider the implementation details, rather solely on the modeling concerns. Information could be represented with the help of relational tables without considering implementation details about the same. However the disadvantage of logical models is the fact that the symbols used to represent the components are flat structures and are in general not intuitive for the modeler to conceive.

Conceptual Information Models : After the introduction of logical models, the need for more expressive solutions to model applications and the respective information bases, was identified. This gave birth to the use of *semantics* in order to model information. Inspired by Cognitive Science [CS88], abstraction mechanisms like generalization, classification and aggregation were used to organize and model information. These models were identified to be more natural and intuitive [HML81]. Therefore, the brief description of the information modeling step in

3.1.1 talks with respect to Entities, their types and relationships, as the conceptual models constitute the state-of-the-art in the field of information modeling for more than four decades.

3.2.2 Modeling Methodologies

The information modeling process is dependent on the modeling methodologies being used to create an Information Model. There are several information modeling methodologies. As per the more recent modeling practices, three underlying methodologies can be identified to build conceptual information models. These are the Entity-Relationship (ER) approach, the functional modeling approach and the Object-Oriented (O-O) modeling approach. There is however a fourth approach, which is capable of capturing the essence of all three aforementioned modeling approaches in a single model - the Graph modeling approach.

The Entity Relationship Approach

The ER approach describes the modeling requirements in terms of entities and relationships between them. It is based on a graphical notation technique. Since its introduction, several extensions to the ER approach have been made available. The building blocks of this approach is the type system represented with the help of entity types, relationship types and attribute types. This approach is intuitive and has been widely used to model real world applications.

The Functional Approach

The functional approach focuses primarily on the specification and decomposition of the functionality of the system. It describes the processes of the system and the information flow between these system processes. This approach uses Data Flow Diagrams (DFD) as a tool to depict the manner in which data is transformed as it flows naturally through the different processes of the system. The DFD is constituted by the processes, data stores, actors and the data flows.

The Object-oriented Approach

The object-oriented approach emphasizes on the identification of objects within the application domain first, followed by their operations and functions. The primary construct here is the object which is constituted of data structures as well as functions. The basic constructs of the O-O model are object classes, attributes, operations and relationships(associations). It is easier to model complex objects with the O-O approach, thereby providing better extensibility and easier integration with programming code.

The Graph Approach

As briefly mentioned earlier, this approach has the capability to encompass all of the three modeling approaches mentioned above. In this type of conceptual information modeling approach, information is modelled with the help of Graphs containing nodes and edges whereby the nodes represent the entities and the edges represent the relationship between those entities. The representation of this information with the help of triples of the form *subject-predicate-object*, where the *subject* and the *object* are the entities of the model and are represented by nodes in the graph. The predicate is represented as an edge between those nodes and corresponds to the relationship between the two nodes (entities). An example of such triples can be seen in the below figure (3.5). Here there are four entities namely *RobotType, Robot, Axis* and a *Workpiece*. This example tries to capture how a graph model can encompass the above three approaches. The *RobotType* can be seen as a class from the O-O approach, the *Robot* is a concrete object of that class thereby has a *isOfType* relationship with the *RobotType* node. The node *Axis* represents an axis of a robot and therefore is connected to the *Robot* node with a *Has* relationship. Similarly to capture what this particular robot does, a separate node *Workpiece* is shown which is connected to the *Robot* node via a *picks* relationship (edge) to show that the robot picks that workpiece. Therefore it can be said that this graph is made of three triples representing four entities and the relationships between them.

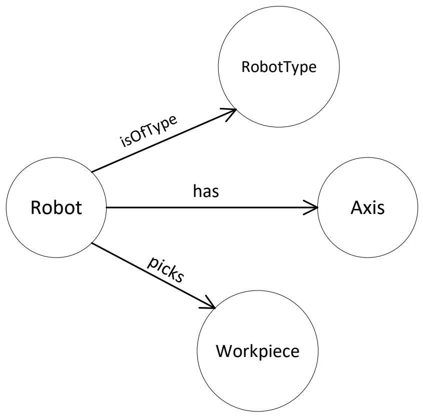


Figure 3.5: An example of a simple Graph Information Model

The decision to choose the modeling methodology is taken right at the start of the modeling process. This decision depends on the modeling purpose and the viewpoint associated with the model. As explained earlier, each information model emphasizes on a different viewpoint of the application scenario, resulting in the possibility of multiple viewpoints of the same scenario. For example, the ER approach needs to be used when the data requirements are at a higher level of detail. In a use-case where functionality takes precedence over the data, the functional approach should be selected. When model extensibility and implementation friendly model is needed, the O-O approach should be recommended. The disadvantages of each of these approaches also needs to be considered. For example the ER approach lacks in the precision to describe detailed levels of data. When there is a possibility of data requirements being changed at the functional level, a model developed as per the functional approach may need to go through major changes. Similarly the main disadvantage of the O-O approach is the way to imagine and deconstruct the system into objects that the system comprises of, where the data and its functions are to be considered as a bundled object. As mentioned earlier, the graph approach is versatile and generic enough to represent any of the other three approaches, therefore it is an ideal tool to explain the core concepts of this thesis and will be used extensively in the forthcoming chapters for a better understanding of the aggregation concept. The biggest advantage of a graph data model is in the speed of information retrieval as compared to relational data models. With the help of query languages like Cypher ([Fra+18]), SPARQL ([DuC13]), GraphQL ([HP18]) etc., which are specifically designed for Graph models, information can be retrieved by running the query on the Graph. The result of these queries is the set of nodes matching the filter criteria of the query. This thesis makes use of SPARQL queries as a means to demonstrate the underlying concepts detailed in the next chapter.

3.2.3 Modeling Steps

In order to develop an information model, in general the steps that need to be followed can be identified as Scope Definition, Requirements analysis, Development and Evaluation as shown in 3.6. These steps have been derived from the information modelling section of the V-Model shown as the grey part in 3.1. All these steps are carried out during the Design and Architecture phases of the integration platform development process 3.1.1. The explanation of each of the individual steps has been detailed in this section.

Scope Definition

The first step in the model development process is the definition of the scope in which the information model will be applicable. This is typically done directly

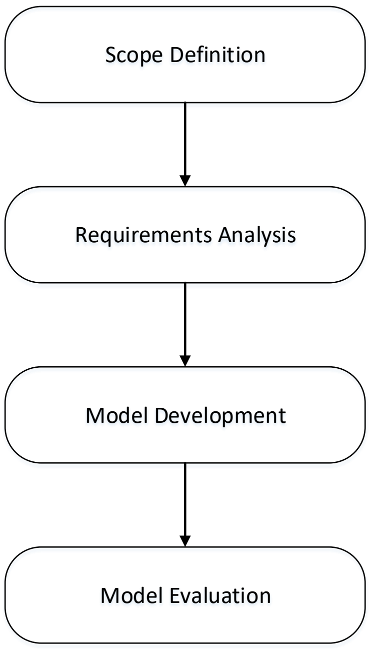


Figure 3.6: Information Model Development Steps

after the requirements are collected for the integration platform 3.1.1. In the four world classification described in 3.2.1, this would directly refer to the usage world. The scope identifies the processes and the domain supported by the information model. It includes the various viewpoints of the model, the manufacturing scenario, the stage of the production life cycle that needs to be supported by the model. The scope statements can be additionally complemented by the activity model and/or the data planning model [TL08]. An activity model depicts the context, data flows and the application processes. High level information requirements can be collected with the help of an activity model. A data planning model focuses specifically on the data requirements and the association between the data components of the model. It is used as a guideline for the identification of interfaces between the various data related components. The scope of an information model can be revisited and revised during the course of information modeling. The scope definition is also used to evaluate the ”completeness” aspect of the information model since it defines the boundaries for the application of the model [TL08].

Requirements Analysis

The definition of the scope is followed by the requirements analysis. The requirement analysis is performed keeping in mind the basic requirements that need to be fulfilled by an information model. The basic requirements can be identified as follows [Myl98]:

- correctness, integrity, consistency, completeness
- low level of complexity through modularity
- clarity and ease of communication
- adequacy as a basis for system development
- provision of a guideline for research

In addition to these basic requirements, the requirement analysis phase should also be complemented by literature research, surveys, domain experts' interviews and assessment of state-of-the-art information models in the application domain. It may also include the best practices and future needs that the model needs to adhere to. The result of the requirements analysis should be documented in a specification document, which can then be used to evaluate the completeness and correctness of the created information model. This document will be taken as a basis for the next step in the development of the information model.

Development of the Model

Based on the detailed scope definition and the requirement analysis specifications, the model enters the development phase. This is the step where the specifications are transformed into a conceptual model 3.2.1. A formal modeling language (e.g. IDEFIX, EXPRESS, UML etc.) is used for the development of the conceptual model and should be independent of the physical implementation of the same. All of the information requirements should be expressed in the conceptual model. This could be achieved with the help of a top-down, bottom-up or a mixed design approach. Although the top-down design approach is considered to be the most effective one for modeling, it may not be be applicable in some cases [TL08]. The design approach is therefore dependent on the individual *usage world*. The basic units (building blocks) of the model are first conceptualized. This is the abstraction process where the requirements are classified into entities, relationships, objects, classes and so on. This is done with the help of the concepts of *Classification, Generalization, Specialization, Aggregation and Association.* Classification as the name suggests is to classify the identified components into data structures and operations and to group them under one umbrella when possible. Generalization, Specialization, Aggregation and Association are the concepts that are used to express relationships between the identified groups or classes. Generalization and Specialization are used to identify *inheritance* relationships between the classes. Aggregation is used to identify *subset and superset* relationships between the classes and finally Association is used to identify *dependency* relationships between these classes. These concepts build the structure of the information model and this structure is then denoted with the help of the appropriate syntax of the formal modeling language used for the modeling.

Evaluation of the Model

After the development phase, the developed information model is evaluated against the facets mentioned in 3.2.3. [Myl98] suggests a comparative framework for conceptual models. Using this framework new conceptual models can be compared and evaluated to identify if a developed conceptual model is appropriate for a particular modeling task. [Myl98] proposes three dimensions along which this evaluation can be made. These dimensions are *Ontologies, Abstraction Mechanisms and Tools.*

- **Ontologies** : List of ontological assumptions made by the conceptual model about the type of applications it aims to model. These assumptions outline the built-in terms of the model and consequently define its range of applications it can model.
- **Abstraction mechanisms** : This deals with the organization of information bases in a conceptual model. The more efficient and intuitive this organization is, the more efficient and effective it will be to search the contents of the information bases thereby making it more extensible.
- **Tools** : The ease of use of tools that perform operations on the information bases, increases the scalability and usability of the information bases.

3.3 A Common Landscape of Information Models in Automation

The discussion in the previous sections have outlined the process of creation and evaluation of information models. Now in today's automation scenario, there exist a plethora of information models pertaining to different viewpoints of the automation system. With the view to standardize information modeling in industrial automation scenarios, several efforts have been made to identify common ground between the innumerable co-exiting information models. This section discusses a few of those efforts keeping in mind the requirements of the Industry 4.0 paradigm.

3.3.1 Reference Models

At this point, it is important to mention the concept of Reference Models. In order to design good quality information models, the modeler relies on pre-defined suitable reference models, based on which the modeler designs the information model specific to the application at hand. [Myl98] refers to such models as “Type 1 Reference Architectures”. Information models thus conceived must be stable, flexible and most importantly extensible, and therefore should be in coherence with existing standards. When an information model satisfies the aforementioned properties, it is then easier to be able to re-use these models and techniques as long as the architectural framework remains the same. There are several reference models pertaining to specific enterprises at specific levels. This property of an information model is called *granularity* [Myl98], higher the level of granularity, the more specific is the reference model. Reference models exist at different levels of granularity. There are reference models which are generic enough to be adapted for various different application scenarios.

3.3.2 Model Universals

Within the domain of automation technology, today there are many co-existing standards outlining basic modeling concepts. Under the umbrella of Industry 4.0, the number of these standards is ever increasing and is now reaching a stage where it is becoming harder to access the standards. The clarity and conciseness demanded by [Ne16a] are not always being adhered to. The need to define a consistent and comprehensive terminology, model worlds and the foundation of common ground between the standards has become crucial. Models like the ‘life-cycle-model’ or the ‘plant-model’ do not deal with well-defined objects but a particular perspective of the objects which is contextual i.e. it can be different for different scenarios. These kinds of models are not independently applicable. [Com16] introduces the concept of Model Universals, which represent such models and provides a referable source to them. A model universal does not model well-defined concrete systems, rather it is a common core of a number of system models. The application domain of model universals is much broader in comparison to reference models. For example any system which has a hierarchical organization will have to comply with the model universal ‘hierarchy’. [Com16] categorizes the different model universals into four broad sections namely, *Processes and Objects* (e.g Entity Model, Life-Cycle Model), *Modeling Schemes* (e.g. ER model, Meta model), *Ordering Structures and Systems* (e.g.Hierarchy Model, View Model) and *Technical Equipment and Functions* (e.g. Component Model, Service Model).

3.3.3 Concept for consistent use of Engineering Models in Automation

In the engineering of automation systems, there exist various information models which have been formulated and are maintained by different groups and trades. For the efficient execution of seamless, tool-supported engineering activities over the plant's life cycle, a homogeneous landscape of models is necessary. [Müh12] identifies this need and develops a method to analyze the models used in engineering of plants. It also established an approach to homogenize these engineering models so as to enable an efficient application of information over the whole life cycle.
[Müh12] analyzes the inter-relationships between the engineering models according to the BMW-principle. It proposes and evaluates based on essential aspects of engineering models in the sense of the digital factory. The result of model classification and analysis is a statement about the similarity of models, so that with respect to a given application, a qualitative statement about transformability or integrability can be given about a tuple of considered models. Building on the findings of model analysis and assessment, [Müh12] introduces the concept of *model profiles*. [DB06] define the term *profile* in the context of fieldbus technology as follows: Profiles are agreements (also referred to as specifications) between device manufacturers as to which parameters with which syntax and meaning between the field devices of a class and the controllers, the parameterization tool or the diagnostics are to be exchanged. The goals of profiles in fieldbus technology are interoperability, interchangeability of devices, comparability, the same behavior, etc. [Müh12] applies the same concept and applies it to engineering models to build model profiles containing a set of model characteristics which can be used to evaluate engineering models. This concept of model profiles was then extended to develop complementary *partial models* to abstract and structure important model components in order to harmonize the existing engineering models. It introduces syntactic and semantic partial models in engineering and explains a method to identify similarities between existing models from a syntactic, semantic and pragmatic point of view. It then further formalizes a method to represent the partial models with ontologies and develops a system to automate the knowledge processing of partial models, thereby developing a common landscape of partial models represented with ontologies.

3.3.4 SemAnz4.0

SemAnz4.0 stands for “Semantische Allianz für Industrie 4.0” (Semantic alliance for the Industry 4.0) [Fay+17]. The aim of this joint collaborative project was the sustainable establishment of German norms and standards as the basis of in-

ternational standardization in the context of Industry 4.0 and thereby creating a basis for investments by industry (both large corporations and SMEs) in products and processes. The project dealt with use cases from different dimensions of the Industry 4.0 paradigm and highlighted the need to formulate standards to develop a *common language* of information exchange between the modeled components.
A typical use case pointed out as a motivation for the project was e.g. when machines independently distribute and redistribute production orders, when sensor data from spatially distributed measurements are merged into a *key performance indicator* at plant management level, and when measuring quality deviations of a partially machined component from production on-line in Design and Simulation Tools are re-calculated, and based on the these calculations other production steps are adapted for this component, then the partners involved must *speak the same language*. The three aforementioned use cases are examples of the dimensions *horizontal integration*, *vertical integration* and *integrated engineering* within the domain of Industry 4.0. "Same language" here means that all parties involved should use terms and "sentence structures" from a shared vocabulary to describe what and how they request or implement something. It is not enough to set up communication channels via interfaces or common communication protocols. The future service orientation also requires a common understanding of content for interacting. Therefore, machines, tools, products, offers, requests and orders must not only be described formally and they can be processed by machines, but also in a way that these descriptions must be known and understood by the designated partner, i.e. all involved must understand the *meaning* being conveyed. For the description of the properties of objects of all kinds, *features* have been proven to be efficient. Features are not only suitable for describing objects, but also (in principle) for describing services. The implementation recommendations of Industry 4.0 describe various use cases [Kag+13]. Each of them interacts with communication partners (people, machines, material, software tools in different, flexible constellations). The prerequisite for realizing these applications is firstly a distributed service-oriented architecture, secondly a communication infrastructure and thirdly a semantic basis so that the communication partners can understand each other.

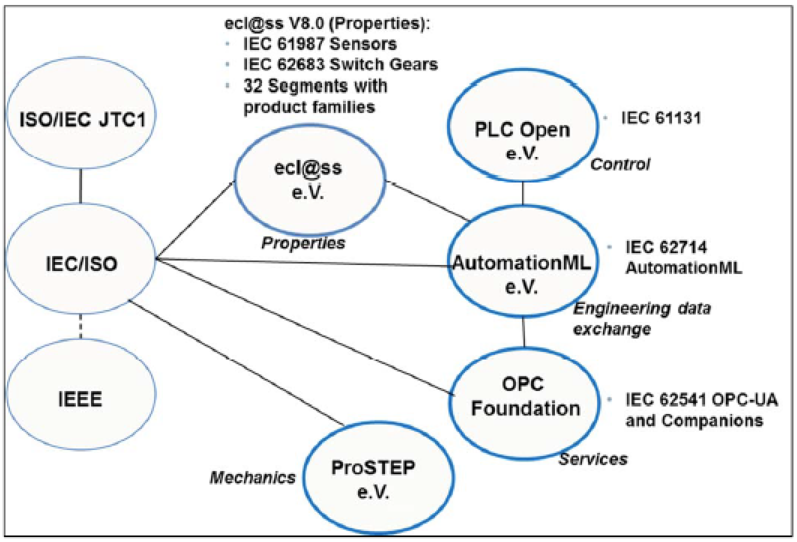


Figure 3.7: Important Organizations and Standardization contributions

SemAnz4.0 refers to the existing feature definitions and tries to combine and structure them accordingly. The analysis of established standards in the context of Industry 4.0 showed that libraries of characteristics ("properties") for essential areas of Industry 4.0 already exist, and fortunately, to a large extent, are the subject of standardization activities.
The existing and established standards for description of components using features (eCl@ss (IEC 61987) in particular) and for structuring information (AutomationML IEC 62714 in particular)) were analyzed and evaluated to form a suitable semantic basis for information exchange in Industry 4.0 applications. For the relevant use cases, the required data models with structure and content were specified. Furthermore, the processes of the use cases were exemplarily demonstrated with software tools.

3.3.5 RAMI 4.0

RAMI 4.0 stands for the Reference Architecture Model Industrie 4.0 [Ne16b]. It is a three-dimensional layered model which brings together the important elements within the domain of Industry 4.0. It has been represented as a three-dimensional coordinate system, with the help of which complex inter-relations between the various aspects of automation can be broken down into smaller and simpler modules and mapped into the coordinate system Figure 3.8. The three axes defined according to RAMI4.0 are as follows:

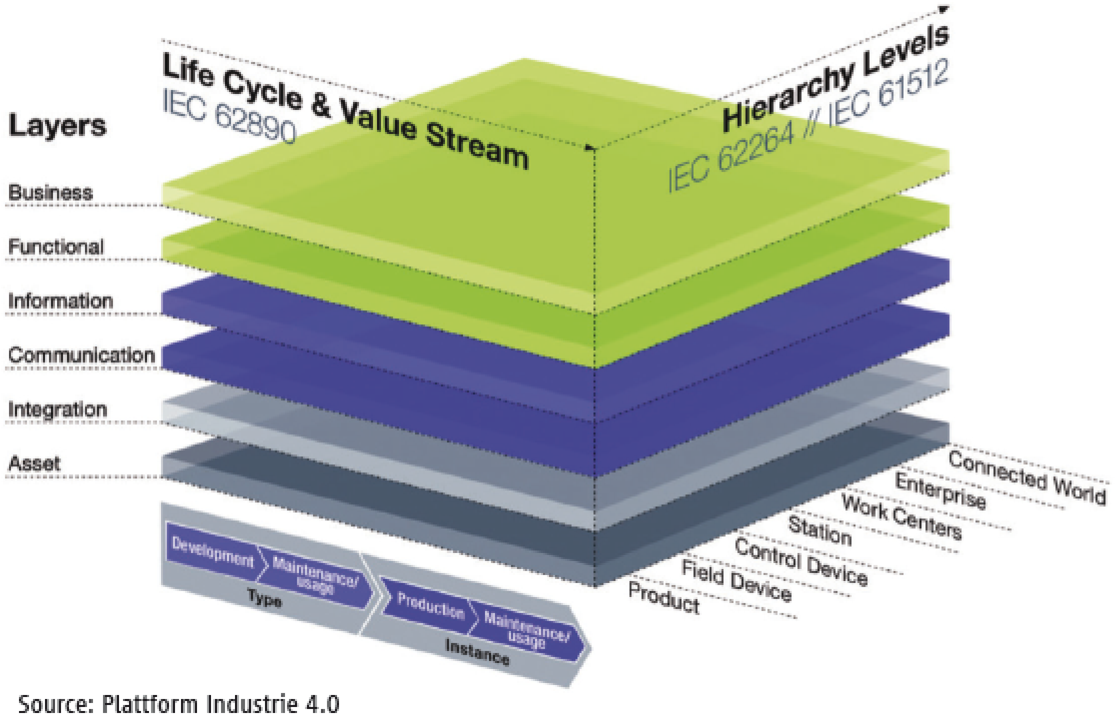


Figure 3.8: Reference Architecture Model Industrie 4.0
[Din]

The Hierarchy Levels axis

The right hand horizontal axis of the coordinate system represents the hierarchy levels of IEC 62264 which is the industry standard for enterprise IT and control systems. The traditional automation pyramid has been supplemented by the levels *Field Device*, *Product* (representing workpieces) and *Connected World* (representing connection to the Internet of Things and Services) in view of the Industry 4.0 environment.

The Life Cycle and Value Stream axis

The left horizontal axis of the coordinate system is used to map the data acquisition along the entire life cycle of production objects in Industry 4.0. This means that even objects that are still under development (referred to as *Types*) can be classified in RAMI4.0. After the designing and prototyping is completed, the *Types* are transformed into *Instances*. The collection and provision of data early on during the product development phase is said to benefit downstream partners in the value network.

The Layers axis

Production objects are recorded along with their data and functions in six layers along the vertical axis. The concept of the Industry 4.0 component plays a key role here. It is the combination of the real production object and its virtual image. These representations are derived from information and communication technologies where properties of systems are represented in the form of layers.
According to RAMI4.0, together these three dimensions map the key aspects of Industry 4.0. It tries to bring together different user perspectives and aims to provide a common understanding of the technologies, standards and use-cases involved. Existing standards and norms can also be classified in RAMI4.0 making it clear if any modifications are needed (e.g. due to overlaps or gaps in the standardization). If multiple standards exist for the same of similar issues, RAMI4.0 provides the possibility to discuss a preferred standard in the reference architecture model. The main aim is to minimize the number of different standards required, thereby moving towards a common landscape for information exchange. Further, RAMI4.0 also highlights the importance of a universal method to uniquely identify components in different information models, unified semantics with a common syntax for data and real-time and time critical requirements of Industry4.0 use cases. It advocates OPC UA as the enabler for Industry 4.0 communication.

3.4 OPC UA - An Enabler for Industry 4.0

In the previous section, the term OPC UA finds mentions during the discussion for SemAnz4.0 and RAMI4.0. It is being increasingly identified as one of the most important enabling technologies for the realization of the Industry4.0 paradigm. This section analyzes the feature set provided by OPC UA which have pushed for its increasing popularity in the last few years.

3.4.1 Motivation and Overview

Open Platform Communication Unified Architecture (OPC UA) was conceived as a successor to the Classic OPC standard, which is still being used in today's automation industries. Classic OPC is based on Microsoft's COM/DCOM architecture and is therefore dependent on Microsoft based platforms. There was no way to use Classic OPC on other platforms and since the COM/DCOM based communication technology is being de-emphasized by Microsoft themselves, there was a need for a new flexible, platform-independent standard which would include all the features provided by Classic OPC and also include a robust data and information modeling methodology. To fulfill the aforementioned requirements OPC UA was created by the OPC Foundation.

OPC UA has a layered architecture as shown in Figure 3.9. The bottommost layer, also known as the *OPC UA Base Layer*, addresses the two basic requirements on the basis of which the standard was created namely *Transport Model* and the *Meta Model*. The *Base Services* layer sits on top of the lowest layer and is also a part of the OPC UA Base Layer. The upper layers consist of the information models based on the OPC UA Base Layer namely the standard information models defined in the OPC UA specifications itself[1],other collaboration models (which will be discussed in a following section) and Vendor specific extensions to the OPC UA information model. The collaboration models can be developed directly on top of the OPC UA Base Information Model or on top of the (Classic) OPC Information Model. Similarly the vendor specific information models can be developed directly using the OPC UA Base, the (Classic) OPC Models or on top of the collaboration models.

3.4.2 OPC UA Base Layer

Transport and Encoding

OPC UA offers different transport and encoding mechanisms depending upon the requirements of data exchange and modeling. Depending upon the use case the following three transport mechanisms have been defined:

1. **SOAP/HTTP Transport** HTTP (Hypertext Transfer Protocol) is the stateless, connectionless request-response protocol for data exchange on the web. SOAP(Simple Object Access Protocol) is a messaging protocol based on XML. OPC UA defines a mechanism to encode the service requests (Read, Write, Method Call etc)/ responses as XML, and transfers the messages with HTTP using the SOAP request/response protocol. This is highly effective in exchanging data and information in Enterprise IT applications which are based on web services and are firewall-friendly.

2. **HTTPS Transport** HTTPS (Hypertext Transport Protocol Secure) is the secured version of the HTTP protocol. In HTTPS all request and response exchanges are encrypted. SOAP is again the messaging protocol used for the

[1]the Data Access (DA), Alarms and Conditions (AC), Historical Access (HA) and Programs(Prg), which encompass the features available in Classic OPC

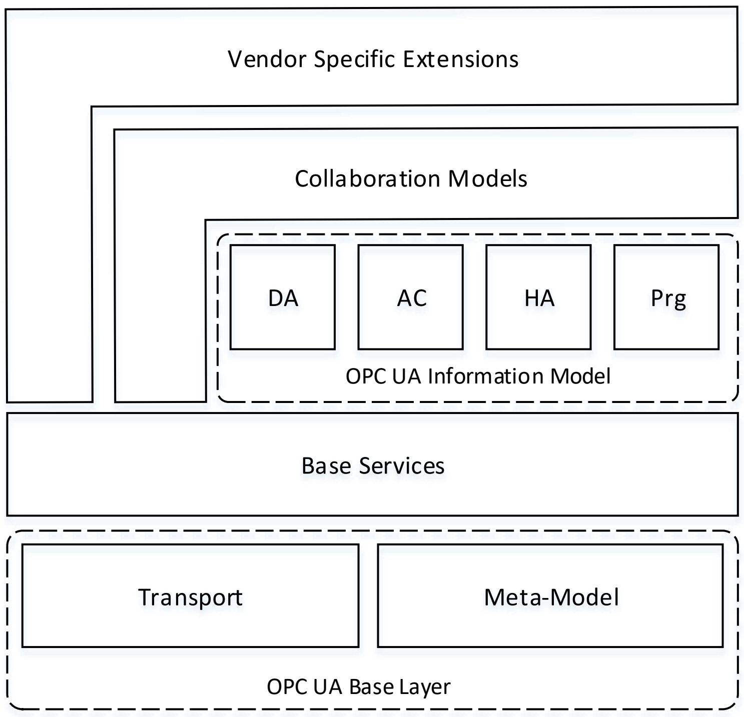


Figure 3.9: OPC UA Layered Architecture
[MLD09] [Fou17]

exchange of messages. OPC UA defines this mechanism for a more secure information interchange.

3. **UA TCP** OPC UA defines this transport protocol alongside Binary encoding for data exchange scenarios where there is a resource limitation, however a fast and high throughput is needed e.g. embedded devices on a factory floor which have a limited memory and processing power. In such a scenario XML encoding is not effective as it is resource heavy as compared to the binary encoding scheme where data is encoded in bits without any additional overhead (as in the case of XML encoding). OPC UA defines a standard TCP based transport protocol for such use-cases. It can be implemented on devices with low resource capabilities.

OPC UA Meta Model

The OPC UA Meta Model defines the rules and the base modeling constructs using which an information model can be created using OPC UA. OPC UA follows an Object oriented approach, at the heart of which are *Types* and *Instances*. The Types define the template of how an Instance of that particular type would look like, what its properties would be and methods that it can call. An OPC UA information model will contain the information about both the types and instances present in the model. The main components of the OPC UA Meta Model have been discussed as follows:

- **Nodes and References**
  All the entities in an OPC UA information model are modeled with the help of *Nodes*. Both, Types and Instances of an information model, are represented as nodes. The relationship between the entities is represented with the help of *References* between the Nodes.
  Nodes have *Attributes* which describe the characteristics of that node e.g. all nodes in an UA information model have an attribute called *NodeId* which uniquely identifies a particular node in the whole address space. Nodes can belong to different *NodeClasses*. The attributes that a particular node will have are defined by the NodeClass.
  References are used to define relationships between the different nodes in an address space. References are instances of *ReferenceTypes*. ReferenceTypes define different kinds of relationships that two nodes can have with each other e.g. a reference of type *hasTypeDefinition* is used to link a particular Instance node to its respective Type Node.

- **Objects Variables and Methods**
  The three main NodeClasses defined in OPC UA are Object, Variable and Method. Nodes that belong to the *Object* NodeClass, are used as the basic structuring element of a UA address space. Objects can contain Variables, Methods and can fire *Events*. Variables are nodes of the NodeClass *Variable* and contain a *Data* attribute which is used to represent the data associated with an Object. The NodeClass Object does not contain a Data attribute, thus any data related to an Object is to be represented with the help of a Variable. The NodeClass Method defines methods related to an Object that can be called. Methods can specify a *Property*[2] called *InputArguments* through which inputs arguments needed for the function to be called can be specified, the value returned (if any) by the method call can be received in

[2]There are two types of Variables defined in OPC UA, namely *Data Variable*, the value of which can change, and *Property* which are used to model the characteristics of an Object and typically do not change during the lifetime of the information model

another Property element of method called *OutputArguments*. The body of the method resides in the UA Server and is executed by the UA Client with the help of the *Call* Service.

- **Base Services**
  OPC UA is a client-server based communication architecture[3](Figure 3.10). A UA Client can access the information model exposed by the UA Server with the help of OPC UA Services. These services have been defined in an abstract manner in the specifications so that they are independent of the transport protocol and the programming environment used to program the UA applications. Some of the important services used by the clients have been outlined below.

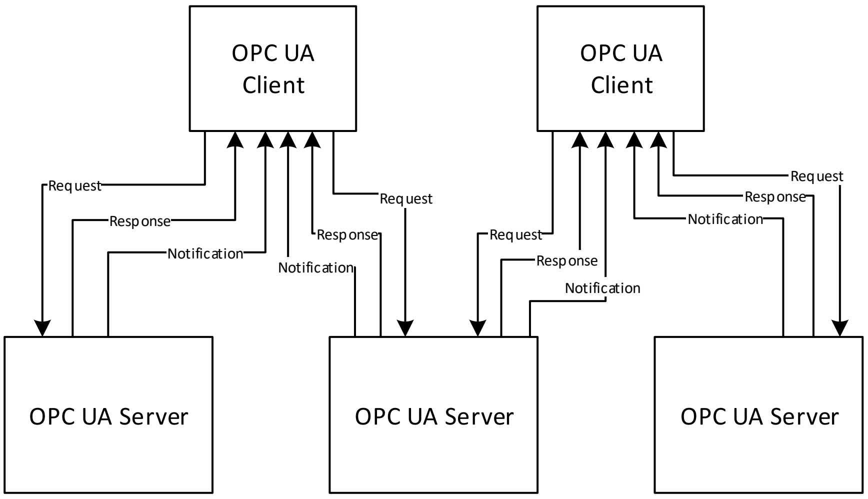


Figure 3.10: The OPC UA Client-Server Relationships

Session and Secure Channel Services Session services are used by a UA client to establish a connection with a UA server. In order to connect with a server, a client must know the *EndpointUrl* of the particular server. The EndpointUrl contains all the required information needed by the client in order to establish a secure connection with the server. The EndpointUrl consists of transport protocol needed, the network address of the server, the security profile needed by the server for a successful connection, any authentication requirements (e.g. username, password) if any and the server

[3]A publish-subscribe architecture for OPC UA was also released in 2018, however for the scope of this thesis report, only the client-server architecture will be discussed.

certificate itself which is used by the client to secure the message exchange. Once the client fulfills all the requirements entailed by the EndpointUrl, it can successfully establish a connection with the server and proceed to access the address space of the server.

Discovery Services The discovery services are used by the clients to *find* available UA servers in the same network. This is specially helpful when a client wants to explore the different connection options provided by the server and can then pick one of them in order to establish a connection with the server. If a UA server chooses to make itself available via discovery mechanisms, it needs to register all of its available connection Endpoints with a *Discovery Server*. The UA client requests the Discovery Server to furnish the available connection endpoints of all the servers that have registered themselves to be discovered. The client then receives a list of available endpoints as a response to its query. The client can then decide to choose one of the endpoints in order to connect to the particular server.

Browse Services Browse services are used by the client in order to find information in a complex address space. It allows the client to transverse through the address space of the server.

Read, Write and Subscribe Services After the connection with a server is successful, a client can use the *Read* and *Write* services to read and write information from/to the address space respectively. If a client wants to monitor a particular node in the address space of the server, it may use the *Subcription* services. When a client subscribes to a particular node it gets notified whenever there are some changes related to that node in the address space.

Node Management Services If a client wants to alter the structure of the server address space by adding, deleting or modifying a particular node, it needs to use the Node Management Services[4].

3.4.3 OPC UA Model Extensions

The layered architecture of OPC UA was briefly discussed in Section 3.4.1. OPC UA is extensible i.e. the specifications are flexible enough to allow vendors and other standardization groups to build their own domain specific information models with OPC UA as the base information model. This is made possible by deriving new models from the OPC UA base information model by extending the base ObjectTypes and ReferenceTypes to formulate domain or vendor specific types.

[4]The server may restrict the alteration of its address space partially or completely. Only if the server allows such changes to its address space structure, can a client do so

This was a result of clear separation between the Address Space Model (Meta Model), the Information Model (derived from the UA Base types and constraints) and the actual Data (Instances which represent the modeled entity). Thus it is possible for a UA Server to expose multiple information models in the same address space.

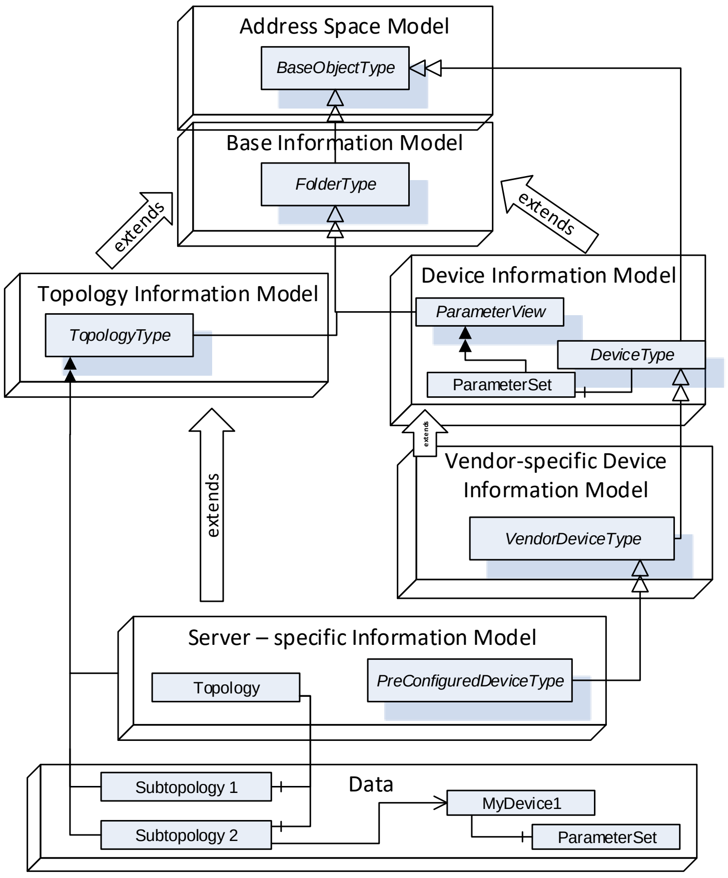


Figure 3.11: OPC UA - An Extensible Model
[MLD09]

Figure 3.11 shows how an address space can be created accommodating several information models. Here the OPC UA Address Space Model defines the *BaseObjectType*. The OPC UA Base Information Model extends the BaseObjectType and creates the *FolderType*. The FolderType is then further extended by the Topology Information Model (into *TopologyType*) and by the Device Information Model (into *ParameterView*). It is always possible to derive types directly from the OPC UA Address Space Model e.g. the *DeviceType* has been derived directly from the BaseObjectType. Then, a Vendor-specific Device Information Model further derives from the DeviceType (defined by the Device Information Model) into a *VendorDeviceType*. A UA Server then exposes a Server-specific Information Model, which in-turn is derived from the Topology Information Model but contains types derived from the Vendor-specific Information Model. Lastly the server instantiates the defined types and exposes its information model in the form of a well-defined address space. This demonstrates the flexibility of OPC UA.

Collaboration and Vendor-specific Models

Because of the flexible information modeling approach, OPC UA has been received very well by the automation industry. OPC UA based standardization efforts have resulted in more than 50 official companion specifications[5], with ongoing efforts on around 15 additional companion standards. These companion specifications cover the information present in all the layers of the automation pyramid across different domains, thus supporting true vertical integration. Figure A.9 in the Annex section shows some of these companion standards at a glance.

3.5 Summary

In today's automation scenario, information integration plays an important role. In the past years however, in light of the Industry 4.0 concept, there has been an increasing demand for vertical integration. This chapter introduces the information integration and information modelling processes and the intricacies of the same. Some of the prominent research projects and works highlighting the importance for a common landscape for information models in the Industry 4.0 narrative were discussed briefly. It was noted that OPC UA has emerged as one of the strongest sets of modeling and communication standards promising seamless vertical integration. The industry has also responded positively and this has resulted in the creation of various OPC UA based companion standards for different sectors and automation levels. These companion standards coexist with a multitude of information models specifically designed with the automation industry in mind inline with the vision of Industry 4.0. This formulates a strong basis for the formulation of an Aggregated Information Model, which could potentially be used to bring the various information models that currently exist under a single umbrella. This challenge and possible solutions will be explored in the forthcoming chapters.

[5]at the time of writing this thesis

Chapter 4

Solution Concept for the Aggregated Integration Platform

Contents

4.1 Basic Concept 56

4.1.1 Overview of the concept idea 56

4.1.2 Method for the solution development 58

4.2 Analysis 59

4.2.1 Actors and Use-Cases 60

4.2.2 Requirements for the Aggregated Integration Platform . 63

4.2.3 Entities in an Aggregated Information Model 65

4.2.4 An Aggregation Scenario from the Process Industry . . 69

4.2.5 Considerations for Information Aggregation 71

4.2.6 Merging of Information Models 73

4.2.7 Demonstration of a Merged Information Model with Graphs 80

4.3 Base Model of Information Aggregation 93

4.3.1 Aggregated Information Model 93

4.3.2 Aggregation Communication Model 95

4.3.3 Consumer Operation Model 97

4.4 Meta-model of Information Aggregation 99

4.4.1 Basic Structure of the Meta-model 100

4.4.2 Data and State Model 102

4.4.3 Basic Methods 103

4.4.4 Addressing Information 104
4.4.5 Communication Sequences 105
4.4.6 Consumer Business Logic 105
4.4.7 Graphical User Interface 106
4.5 Summary 107

4.1 Basic Concept

This section describes the basic concept for the creation of an *Aggregated Integration Platform*. The section 4.1.1 gives an overview of the basic concepts involved in the development of the integration platform. The subsequent section 4.1.2 details the methodical approach for the development of the solution.

4.1.1 Overview of the concept idea

The discussion in Chapter 3 had concluded that an efficient integration platform exposes an information model, and provides easy access to the data and functionality provided by the underlying system for which the information model is designed. Due to the rapid advent of technology, devices and subsystems in automation are now capable of exposing their own information models, thus advancing the trend towards distribution further. The main aim of an *Aggregated Integration Platform* is to assimilate the multiple distributed information models and to expose them via an *Aggregated Information Model (AIM)*. The aggregation of the *UIMs* will be done in such a way that the aggregated model would still provide the same data and functionalities that the individual UIMs provide.
The aggregated integration platform will consolidate other UIMs. This means that it will collect the data and the functionalities from the UIMs, organize them, and make them available to the consumers. The consumer applications will be able to select the kind of information they need from the aggregated information model, depending upon their specific use cases. Since the data and information about multiple systems will be made available by the aggregated integration platform, this gives the possibility to the consumer applications to gain access to the information from multiple sources but consolidated in a single source of information, *all in one place.*
The access and security aspects shall be automatically handled by the aggregated integration platform and shall not be compromised. In automation systems, there are basically two types of data and information consumers, namely People (human beings) and other technical systems. These consumers are further divided

depending upon priority and field of operation. It is important for the aggregated platform to control access to the data and information it exposes. Since an Aggregated platform will contain information from multiple sources, the consumer's access rights is of primary importance. The efficient management of access rights in the aggregation platform will safeguard the data and information accessible via the aggregated platform. Furthermore, the aggregated platform will also aggregate the security and access requirements already provided by the underlying information sources. This means that if in order to access the information from an underlying source (directly), a consumer needs a specific security and access control profile, it will also be the same while accessing information via the Aggregated integration platform (Figure 4.1).

The Aggregated platform will act as a layer of abstraction between the con-

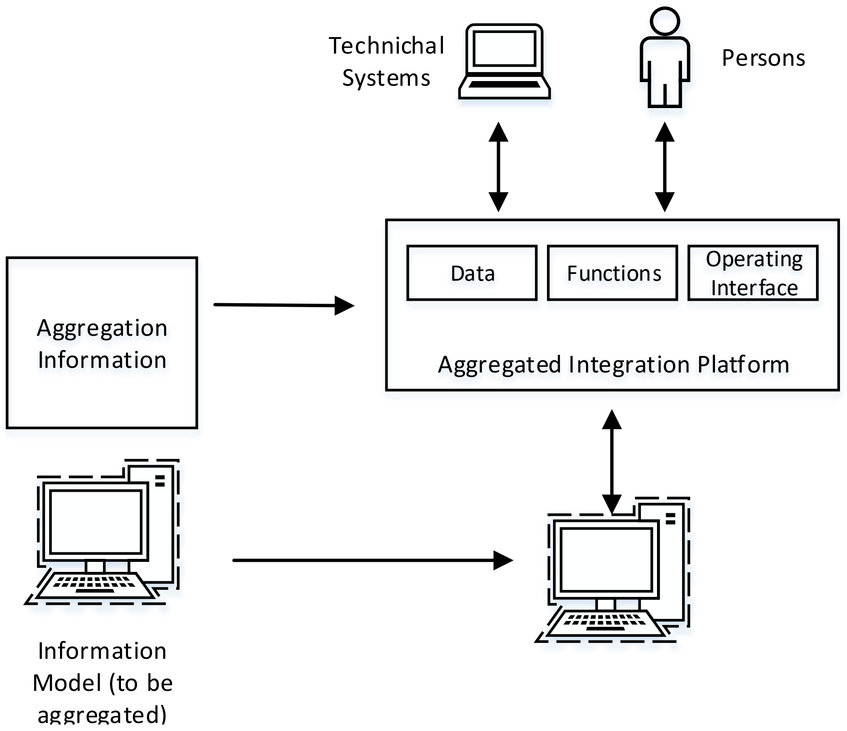


Figure 4.1: Basic Concept of the Aggregation Platform

sumers and the sources of information. The communication mechanisms used by the aggregation layer to communicate with the underlying sources can be different. However, to the consumer the underlying communication mechanisms will be irrelevant as it will just know about the single communication mechanism with the Aggregated platform. For the consumers, the Aggregated platform will be the single source for all data and information.

4.1.2 Method for the solution development

In order to develop the aggregated integration platform, the V-Model introduced in 3.1.1 was closely followed as part of this thesis. During the design and architecture phase of the V-Model, the base model, meta-model and the architecture of the platform was created. The following subsection outlines the procedure followed for the development of the meta-model. Section 5.1.2 describes the procedure for the development of the architecture of the aggregated integration platform.

Procedure for the meta-model development

The procedure for the meta-model development (4.2) closely follows the standard Software-Development procedure [BD10]. An important step for the solution development is the definition of the *Requirements* for the development of the solution. In order to define the requirements, a *Requirement Elicitation* phase was conducted. In this phase, the *Actors* expected to use the solution, and the *Use-Cases* where the solution will be used, were identified. At the same time, a *Review* of existing information models used in the automation industry was conducted to identify the important points covered while formulating the specifications of such an information model. At the end of the analysis phase, the basic requirements for the Aggregated integration platform were obtained. The requirements were classified as functional and non-functional requirements.

After the requirements were frozen, the Analysis phase was conducted. The objective of this phase was to formulate the Base Model of the aggregation platform, where the different objects of the aggregation system were discussed and structured, thus resulting in the Base object model, which is also an outcome of the analysis phase [BD10]. Upon refining the base model with further concrete details gathered from the analysis of the existing information models, the meta-model of the aggregation system is realized. The meta-model is defined through abstraction, generalization and structuring of the modeled information sources of the application domains. The graphical representation of the Meta Models with the help of UML-diagrams is suitable in order to effectively show the relationships between the elements of a complex system.
The base model is an informal model which is made up of layers in a functional hierarchy. These layers are conceptualized in a way that the elements within a layer are tightly coupled to each other, however the individual layers are as loosely coupled (to the other layers) as possible. The meta model is derived from the base model and is intended to be used to aggregate any type of information model

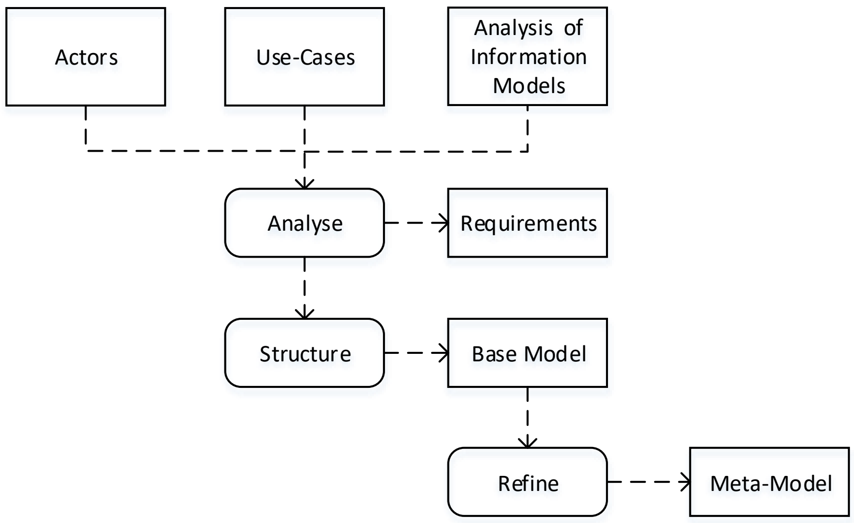


Figure 4.2: Procedure for the meta-model development

in the aggregated space and this is the biggest challenge. This meta-model for the aggregation of information models serves as the base for the development of the architecture of the aggregated integration platform in Chapter 5 and for the implementation of the same in Chapter 6.

4.2 Analysis

The Analysis phase observes the *aggregation platform* from a functional point of view. This approach looks at the *role and capabilities* of the aggregation platform. Therefore it is important to outline the *Actors* who interact with the aggregation platform and the *Use-Cases* that provide the basis for such an interaction. The use-cases define which actor performs what role during their interaction with the platform. The following subsections discusses the actors and the Use-cases who would be involved within the framework of *Information Model-Management*.

Definition 4.1 ***Information Model-Management***
Information Model-Management encompasses all the activities in an automation life cycle that are involved with the information models which co-exist in an automation environment. Therefore, accordingly, it includes the functions that are required in order to carry out these activities. These functions are not necessarily

implemented by a single system but can be distributed among several sub-systems with a specific focus towards the information model.

4.2.1 Actors and Use-Cases

Actors

Figure 4.3 shows the actors who are involved in any capacity with Information Model-Management. The definition of these actors is based on their respective roles in the plant-life cycle. The actor called *System* in Figure 4.3 can actually be considered important for any phase in the plant-life cycle and therefore has been associated with every phase in this figure. The description of the individual roles of the actors (that have an impact on Information-Model Management) has been outlined in the Table 4.1.

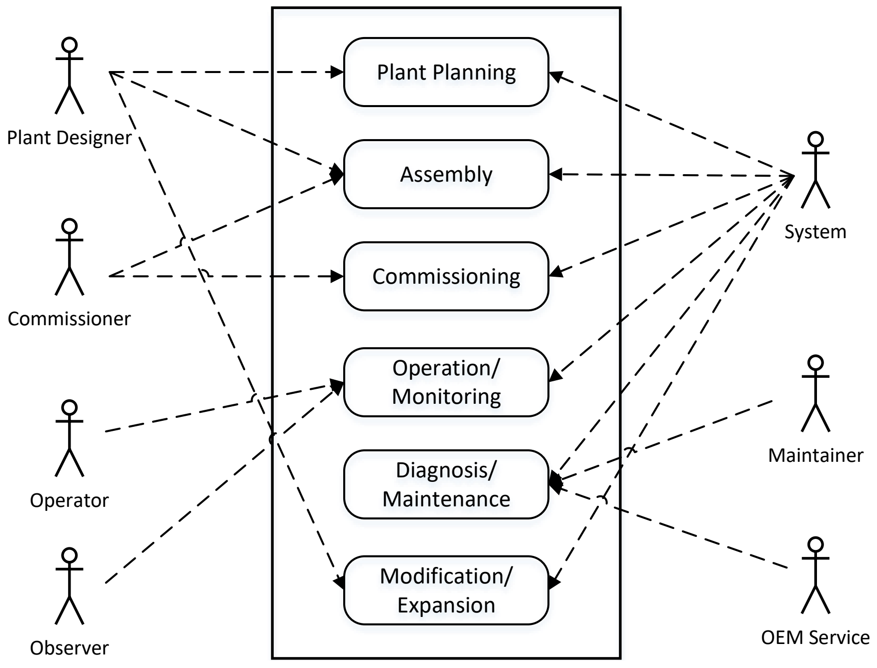


Figure 4.3: Actors

Use-Cases

The use-cases describe how the actors interact with the system. Figure 4.4 shows the use-cases for the system *Aggregation Platform* as well as the actors that inter-

Actor	**Description**
Plant Designer	Planning of the Topology and the devices involved in the engineering system Basic configuration and communication configuration
Commissioning Engineer	Commissioning of the automation components including their communication Execution of the function testing and fine parameterization of the components
Operator	Operation of the Production System Installation of the process relevant parameters
Observer	Observation of the Production system's functioning Monitoring of the Status messages of the individual sub-systems
Maintenance Engineer	Maintenance and Repair of the components of the system Carrying out system diagnostics
OEM Service	Carrying out vendor specific operations
System	The system that uses the Aggregated Integrated Platform (e.g. an application that implements an Information-Model Management function)

Table 4.1: Description of Actors

act with it. An actor is usually associated with multiple use-cases. The figure does not depict which actor is associated with which use-case for the sake of simplicity. The conceptualization of the use-cases was done through the analysis of several sources [Fou21b] [Fou22] [Gro+13] [Fou21c] [Kle+17a] [DC+19] [OF21] [OF19b] [OF19a] [Bou+21]. Table 4.2 describes the Use-Cases shown in the figure. It is important to note here that in the case of Information-Model Management, a particular Use-Case can involve multiple Actors depending upon the modeling criteria and purpose. The focus of this table is to showcase the usage of the *Aggregated Integration Platform* by the Actors in a distributed environment, therefore other Use-Cases considering other aspects of the automation system have not been considered. These Use-Cases form the basis for the Requirements Elicitation phase, which are covered in the following section.

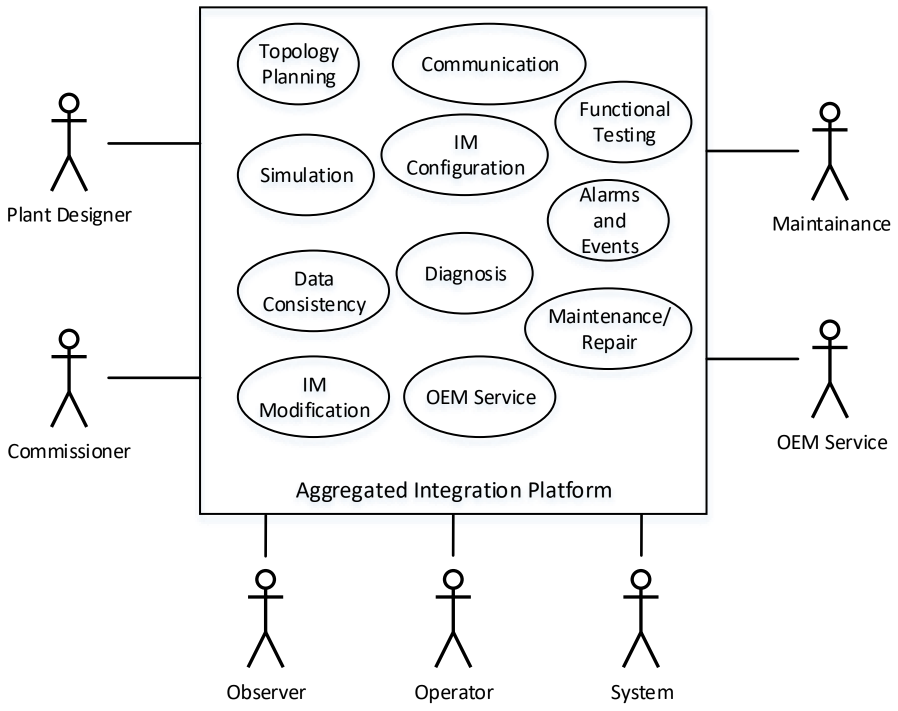


Figure 4.4: Use-Cases

Use-Case	**Description**
Topology Planning	Determination of the components and their topologies in their respective information models on the shop floor
Communication	Determination of the communication specific information e.g., protocol, addressing scheme for the UIMs
Simulation	Usage of AIMs for simulated components enabling virtual commissioning
IM Configuration	Activities related to the configuration of information models e.g. user access rights, security profiles etc.
Functional Testing	Testing of the functionality and services provided by the AIMs
Alarms and Events	Provision of alarms and events by the AIMs depending upon conditions established during the UIM design
Data Consistency	Maintaining the consistency of data between an AIM and the respective UIMs
Diagnosis	Detection of anomalies based on the collective information of the UIMs and their respective dependencies
Maintenance/Repair	Carrying out maintenance/repair activities on the UIMs or on the underlying sources themselves
IM Modification	Modification of the UIMs directly from the AIM e.g., addition, deletion, updation of an UIM element
OEM Service	Carrying out vendor-specific operations on the information models

Table 4.2: Description of the Use-Cases

4.2.2 Requirements for the Aggregated Integration Platform

This section outlines the requirements for the aggregated integration platform. The requirements are derived from the use-cases defined in Section 4.2.1. Requirements signify the set of features that are necessary for a system to possess. They can be of two types functional and non-functional requirements. Functional requirements define the functionality that a system must provide. Non-functional requirements are those features of a system that not directly related to the functionality of the system but still need to be fulfilled [BD10](e.g. response time of a request-response system). The functional and the non-functional requirements have been outlined in the following tables (Tables 4.3, 4.4).

Requirement	**Description**
User Roles	Support for different User roles and access rights
Completeness	Representation of all Information Model-Management relevant artifacts
Handling Underlying Models	Addition, Modification and Removal of new Information Models
Information Access	Access and explore the UIMs
Function Support	Access and execute the functions implemented in the underlying systems, directly from the Aggregated platform
Read/Write Data	Communication with the data and information sources and read and write data from/to them e.g a functionality provided by the underlying system should also be provided by the aggregated platform as a proxy
Subscriptions	Subscribe to data points of underlying system for automatic updates of value changes
Online/Offline	Access to the information both if the source is in offline or online mode
Alarms	The alarms and events supported by the underlying systems should also be supported via the aggregated platform
Handle Mapping Rulesets	Import or Export Instance and Type Mapping Rulesets for the particular information model.
Semantic	Organize the data and information about the same resource modeled by different information models

Table 4.3: Functional Requirements

In conclusion, it can be said that collection of use-cases and requirements was done from the point of view of the aggregated model, keeping in mind at the same time the requirements fulfilled by the individual underlying models. In essence, the requirements and use-cases common to all the underlying models should be addressed by the aggregated model.

Requirement	**Description**
Platform-Independence	Independent of Operating System platforms (see 2.3.3)
Robust	The aggregated integration platform represents a central layer of abstraction which aggregates the information from other information systems. Therefore there should be no characteristics which could threaten the robustness of the entire system
Domain-Independence	Independent of the application domain (see 2.3.3)
Openness	Open for information models from multiple vendors and players (see 2.3.3)
Transparent	Communication with underlying systems should be transparent(see 2.3.3)
Multiclient	Multiple clients should be able to connect to the platform at the same time and access data and information from the underlying systems
Multisource	Multiple underlying information sources should be accessible from a single or multiple consumers (clients) at the same time
Secure	The communication between the integration platform and the underlying sources should be secured with the help of certificates

Table 4.4: Non-functional Requirements

4.2.3 Entities in an Aggregated Information Model

As discussed in section 4.1.2, the objective of this chapter is to develop a meta model for the aggregated integration platform. This was done on the basis of an analysis of the different existing information models in use today in the industry. The information models that were analysed were briefly discussed in the previous chapter (Section 3.4.3). This section describes the result of the analysis of the information models to outline the commonalities between them in order to identify the required entities for Information-Model Management.

The below mentioned discussion consists of the basic understanding of the entities in an object oriented information model[1] and the relevance of those entities for the so-called aggregated information model i.e. how those entities should be handled by the aggregated information model.

- **Object Types**
  Depending upon the modeling purpose, the specifications define *Object Types* which define a template for the different types of components that could be conceptualized. This can be equated to *Classes* in *Object-Oriented Design* terminology.

[1]non object-oriented modeling approaches have not been considered within the scope of this thesis as they would require an entirely different aggregation methodology

Relevance for the Aggregated Information Model The Instance/Object Types of the aggregated information model should be a collection of all the Instance/Object[2] types defined by the UIMs being aggregated. In addition to that the Aggregated Information Model could also define its own types which do not belong to any underlying model.

- **Objects**
  When a component in a system is being modeled, it is represented as an instantiation of these well-defined Object types (mentioned above) and is also known as a *Digital Representation* of the component. The definition of the term within the context of this thesis is given below.

  Definition 4.2 ***Digital Representation***
  A Digital Representation is the representation of a component of a system in the cyber domain. It serves as a layer of abstraction between the actual component and the services that want to access the data and information from the component (Figure 4.5). It is a part of the middle-ware which constitutes the interfaces required to establish communication with the actual component.

  Figure 4.5 shows three *real* components Component 1, Component 2 and Component 3 being represented by their respective digital representations namely DR-Component 1, DR-Component 2 and DR-Component 3 within an information model. All communication with the real components are achieved via the digital representations of the same.

  Relevance for the Aggregated Information Model The instances representing the system-components (Digital Representations) in the Aggregated Information Model should be a collection of the instances represented in all the underlying models. Similar to the case of the Object-Types, the Aggregated Information Model can also contain its own Objects which do not belong to any underlying model i.e. additional objects to the ones defined in the underlying model but represented only at the aggregated level e.g., an object in the aggregated model representing the mean energy consumption of a system in a scenario where the aggregated model aggregates energy related information from several underlying models.

[2]the terms ‘instance’ and ‘object’ have been used interchangeably throughout the thesis to mean instatiation of the defined types

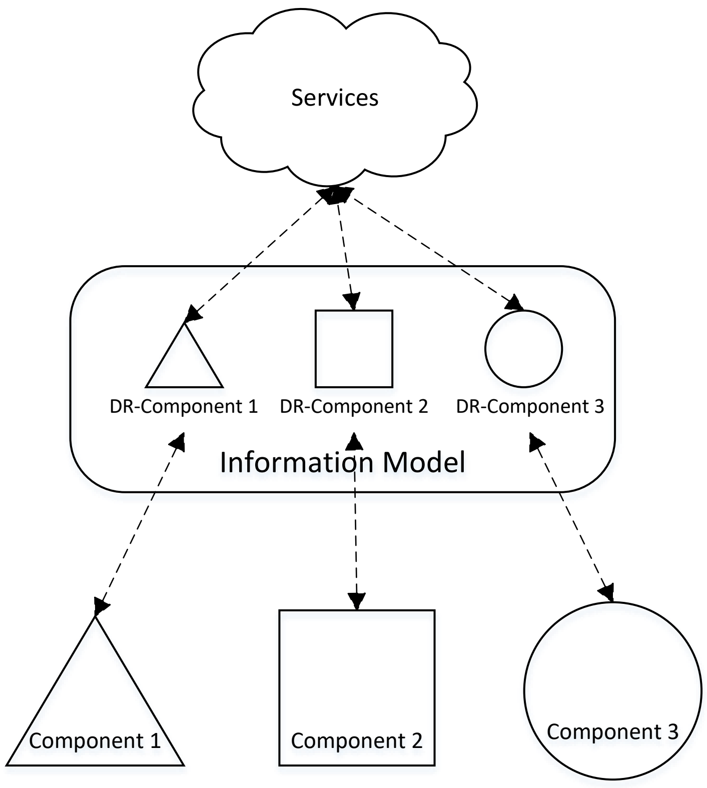


Figure 4.5: The Digital-Representation Concept

- **Relationships and Relationship Types**
  An information model also defines the *relationships* between two types, between a type and an object or between two objects. In *Object-Oriented* terminology, these relationships can be represented as instantiations (objects) of well-defined *Relationship-Types* (classes).

  Relevance for the Aggregated Information Model The Relationships and the Relationship-Types of the Aggregated Information Model should be formed by the union of all the Relationships and Relationship-Types defined by the UIMs (that are being aggregated). The aggregated information model may also define its own Relationships and Relationship-Types which do not belong to any of the underlying models.

- **Functions**

Functions (also called Methods) define the functionality that can be carried out by the *Digital Representations* of the modeled components. In *Object-Oriented* terminology it can be represented as a method that can be called by the object upon providing the appropriate input arguments and the result can be returned by the method.

Relevance for the Aggregated Information Model The Aggregated Information Model should contain all the functions/methods provided by the underlying models. When such a function is called by a service the function call should be forwarded to the underlying system and the objects returned (if any) by the underlying system should be forwarded back to the requester. This means that the implementation of the function would still be in the underlying system and not the Aggregated platform. In other words the aggregated model should provide a 1:1 mapping from the aggregated method to the underlying method to be called when the aggregated method is called.

- **Alarms and Alarm Types**
  Information Models define Alarm Types which provide a template for the types of alarms that can be modeled in a system (dependent on the modeling purpose). They define the predefined conditions necessary for a particular *Alarm* to be thrown. When these conditions are satisfied, the alarm is triggered. Alarms are the instantiations of these Alarm Types.

  Relevance for the Aggregated Information Model The Aggregated Information Model should contain all the alarms and their respective types that the underlying models define. As in the earlier cases, the Aggregated Information Model can also define its own Alarms and their types, which do not belong to any underlying system. When an alarm is generated in any of the underlying systems, it should be propagated to the Aggregated integration platform and in-turn to the end consumer application expecting the alarm.

- **Events and Event Types**
  Information Models define Event Types which provide a template (class) for the types of events that can be modeled in a system (dependent on the modeling purpose). They define the predefined conditions necessary for a particular *Event* to occur. When these conditions are satisfied, the event is triggered. Events are the instantiations of these Event Types.

Relevance for the Aggregated Information Model The Aggregated Information Model should contain all the events and their respective types that the underlying models define. As in the earlier cases, the Aggregated Information Model can also define its own events and their types, which do not belong to any underlying system. When an event is generated in any of the underlying systems, it should be propagated to the Aggregated integration platform and in-turn to the end consumer application expecting the event.

4.2.4 An Aggregation Scenario from the Process Industry

Figure 4.6 shows a scenario where three different information models based on industrial standards from the process industry namely DEXPI (Data Exchange in the Process Industry), PA-DIM (Process Automation Device Information Model) and OPC 40223 (OPC UA for Pumps and Vacuum Pumps) describe the same physical device, in this case an industrial Pump. The three different information models look at the Pump with different modeling purposes in mind. The DEXPI model defines a data model for the pump facilitating the modelling and transfer of the engineering information of its P&IDs (Piping and Instrumentation Diagrams) based on the ISO 15926 standard. The PA-DIM information model is based on the NAMUR (User Association of Automation Technology in Process Industries) requirements for Open Architecture (NE 175, also known as the NOA concept), Self monitoring and Diagnosis of Field Devices (NE 107) and NAMUR standard device - Field devices for standard applications (NE 131).

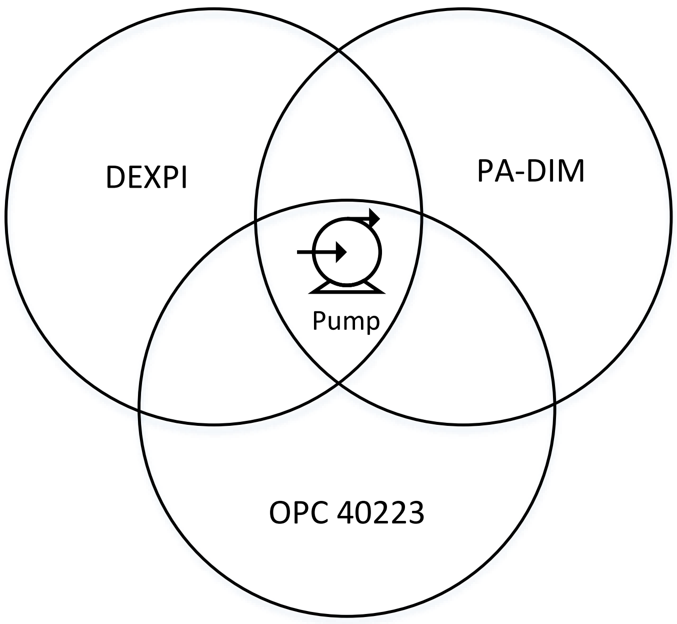


Figure 4.6: An Aggregation Scenario from the Process Industry

For the industrial pump in question, the corresponding PA-DIM model would focus on providing standardized information for Identification, Diagnostics, Process Values and Configuration parameters. Similarly OPC 40223 specifies a manufacturer independent Asset Administration Shell (AAS) for pumps and vacuum pumps for applications in various industries including the process industry. This would mean that in our scenario, the OPC 40223 information model would provide an AAS submodel [WT21] specific for the pump in question, thus enabling the support for continuous and dynamic engineering over its lifecycle. Now consider a consumer application which needs to access the information about this pump from all three information models at the same time in order to cater to some predefined use-case. In such a case, it would have to connect to the three information sources individually and assimilate and process the information on its own. This is where Aggregation could be useful. The aggregated information model will aggregate all the information about the same physical entity (in this case the pump) from the three different UIMs and organize it in such a way that the consumer application should be able to access it by just connecting to the aggregated platform. The aggregated platform will then act as a gateway to the information from the three underlying information models, but the consumer gets access to the information *all-in-one-place*. The challenges in such an aggregation scenario will be presented

in a structured manner in the forthcoming section.
The creation of the aggregated information model for this scenario has been exemplified with the help of Graph models for the pump corresponding these three information models. As mentioned in the previous chapter, SPARQL queries will be used to demonstrate how this can be achieved.

4.2.5 Considerations for Information Aggregation

Following the discussion of the above aggregation scenario 4.2.4, a few considerations about aggregation of information models need to be taken into account. As per the previous discussions about aggregation, it can be concluded that in its most basic form, an aggregation of information models would mean to bring together or *aggregate* all the *Types* and *Objects* (modeled by the information models[3] being aggregated) into a single aggregated information model. The way the Types and Objects will be aggregated will depend on the information models being aggregated. This can be explained with the help of the following Table 4.5.

	Same	**Overlapping**	**Distinct**
Types	Nothing	Merge	Add
Objects	Nothing	Merge	Add

Table 4.5: Aggregation Scenarios

The aggregation scenarios (shown in the table) can be explained as follows. For better understanding, at the beginning of the aggregation process, the aggregated information model can be considered to be empty i.e. without any *Types* or *Objects*. The assumption here is that the UIMs are being aggregated one after the other in succession i.e. at first only one underlying model will be aggregated by the aggregating platform, then the second, then the third and so on. When the aggregating platform starts to aggregate a new underlying information model, the underlying model will be compared to the already existing aggregated information model (formed as a result of previous aggregation cycles).
Now let us look into the composition of the aggregation process to understand how exactly should a new UIM be aggregated by the aggregation platform. For this, let us consider the *Types* first and discuss the corresponding three cases outlined in the table above 4.5.After the discussion about the aggregation of *Types*, the same three cases will be discussed for the aggregation of *Objects*. In addition to this, the *Relationships* between *Objects*, between *Objects* and their *Types*, between

[3]since all models are object oriented, it can be assumed that all three models define Types and Objects

Types shall all be preserved in the aggregated model. The following section considers these *Relationships* to be instances of *RelationshipTypes*, therefore they can also be considered to be covered by the following discussion, and have not been explicitly mentioned.

- **Aggregation of Types**
  Do Nothing In this case, if the information model being aggregated is based upon the same modeling rules and conventions as an information model which has already been aggregated (e.g. trying to aggregate an PA-DIM based information model, when another PA-DIM based information model has already been aggregated by the aggregating platform), the *Types* modeled by the *new* information model are already present in the aggregated system (because of a previous aggregation cycle), therefore nothing new has to be done for the types.
  Merge If the new information model (being aggregated) has an overlap with an already aggregated model with respect to *Types* (semantically similar types), then the new *Type* should be merged with the already aggregated *Type* and a mapping should be maintained between the original underlying *Type* and the *merged* Type in the aggregated model. This case will be clearer to understand in the forthcoming section when mapping rules will be discussed in a greater detail in 4.2.6.
  Add If the new information model is totally distinct from any of the already aggregated models i.e. if it is a model based on different modeling rules and conventions that any of the ones already aggregated by the aggregating platform, then the new type needs to be *added* into the type system of the aggregated model (e.g. when the aggregating platform is aggregating an PA-DIM based information model for the first time, without having aggregated any PA-DIM based information model in any of the previous aggregation cycles).

- **Aggregation of Objects**
  In case of instances, the same three cases exist, however it is independent of the information model being aggregated.
  Do Nothing When aggregating a new information model, if the exact same information about the same physical entity has already been aggregated via another information model then it should not be duplicated, so nothing needs to be done for this particular digital representation of the physical entity. For example if 2 information models based on PA-DIM, model the same physical device (overlapping information models based on the same standard), then both information models will contain exactly the same information about the device. So in this case while aggregating the second information model,

the object representing the device can be ignored so as not to duplicate the device information.
Merge This is the case similar to the one described in the example scenario in 4.2.4. When different aspects of the same physical entity are described by information models based on different modeling rules and conventions, then the information about the entity from all the underlying information models, should be merged within a single digital representation of the entity within the aggregated information model.
Add In the case of objects representing physical entities, this case is relatively simple. While aggregating an object in an information model, if the object (digital representation) represents an entirely different physical entity than the ones that have already been represented in the aggregated information model (during previous aggregation cycles), then this object representing the *new* physical entity should be added to the aggregated information model.

4.2.6 Merging of Information Models

As discussed in the previous subsection, when aggregating information models, there could be a need to merge *Types and Objects*. This means that there needs to be a mechanism to understand if two objects or types are semantically the same or similar. The main challenge is in the case of objects i.e. how do we understand if two or more digital representations (objects) are actually representing the same physical entity? This section tries to analyse if a similar problem had been solved in other domains of information modeling and discusses the possible solutions that can be considered in the case of our aggregation scenario.
A similar challenge was encountered in case of the *Common Information Model* (CIM) instrumentation services. The CIM is an open standard for IT environments managed by the Distributed Management Task Force (DMTF) ([DMT12]). It is used to represent the managed elements in an IT environment as a set of objects, and relationships between those objects. These representations are manufacturer or provider independent, which enables multiple applications to seamlessly exchange information about the modeled elements. CIM also provides the architecture to manage and control these elements. Having a common model of information enables management applications to communicate without costly conversion or information loss. The WBEM (Web-Based Enterprise Management) is one such CIM implementation which includes the discovery and access protocols for other such CIM implementations. In CIM too, the same entity/type can be represented by different instrumentation services and an automated client is then used to determine whether the different services are representing the same entity with the help of *correlatable identifiers*. Depending on the entity, correlatable identifiers are attributes or properties like *SerialNumber*, *GUID* (Globally

Unique Identifier), *MAC Address*(Media Access Control address), *WWN* (World Wide Name), WWID (World Wide Identifier) or other properties that have the same value across the different services. In a newer REST based management interface called Redfish, different Redfish instrumentation services can actually be interlocked using URIs (Uniform Resource Identifier) that link to each other and a single entity representation can be referred to. However, in Redfish as in CIM or other management interfaces, using correlatable identifiers is the more typical mechanism to represent the same entity in multiple services [Usl+12].
However it is impractical to use correlatable identifiers in order to solve the challenge of merging types and objects in the aggregation scenario in case of automation systems. This is because different information models are defined for different modeling purposes and proposing this as a solution would mean that the correlatable identifiers should be included as a part of the modeling standards. This has to be done for each entity modeled in the information model. This solution, albeit straightforward, will cost a lot in terms of effort and time to be able to implement it on a global scale. Secondly any new identifier being assigned to an entity or a type has to be cross-checked with a global storage of ids in order to make sure that the same id is not provided to any other modeled entity or type. This in itself is an implementation challenge.
ISO/TS 29002 Part 5 does introduce an identifier called "international registration data identifier" (IRDI) for an administered item [ISO09]. However, this administered item has been explained to represent a concept or concept information element in a concept dictionary. Moreover, it has been developed to manage terminologies (terms, definitions, images etc.). The concept as such can be applied to our scenario indeed but it would again result in the same challenges as discussed above. The usage of interlocking entities or types with the help of URIs would also not be useful in our case, again because of implementation issues. Furthermore, a straightforward logic to compare global identifiers means that such a global identifier should already exist as a part of the available standards in the automation industry today, which unfortunately does not. Thus, it needs to be kept in mind that the suggested solution should build on top of the existing standards without the need to change the contents of already existing standards.

Contextual Instance and Type Mapping Rules

The solution proposed in this thesis is to solve the challenge with the help of *Contextual Mapping Rules* for objects and types. The *Instance Mapping Rules*(IMR) should define how an instance represented in an information model *maps* to another instance in another information model. Similarly the *Type Mapping Rules*(TMR) maps two types in two different information models. For example, for two information models A and B (Figure 4.7), *IMR-AB* and *TMR-AB* define the rules according to which an instance/type from information model A should be considered to be the same/similar to an instance/type from information model B respectively. If according to the mapping rules two different instances/types are deemed to be the same then they should be merged into a single instance/type respectively. When a new information model is being aggregated, its types and instances should be checked against these mapping rules to aggregate them properly. The usage of such rules will be discussed in a greater detail in Chapter 5.

In order to merge a particular instance/type coming from two different informa-

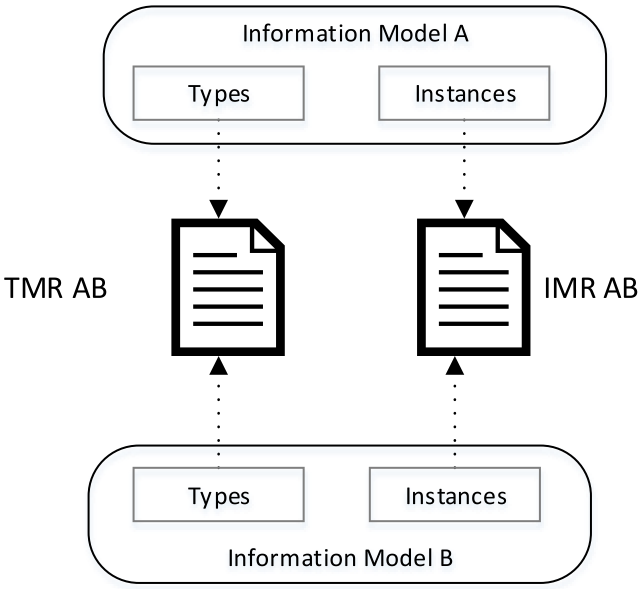
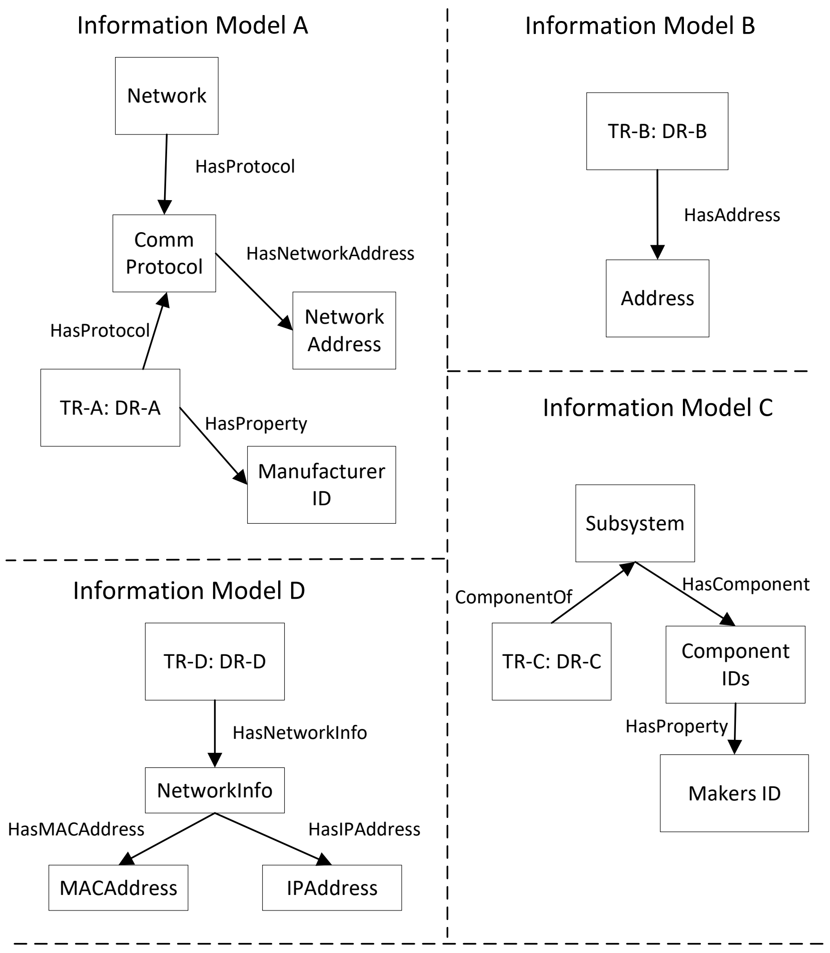


Figure 4.7: Instance and Type Mapping Rules

tion models into one, there needs to a comparison between one or many properties of that particular instance/type in the two information models. When the result of all the respective comparisons shows that they are indeed two representations of essentially the same entity in the two information models, then these two representations should be merged into one instance/type in the aggregated information model. The properties to be compared are described in a so-called *mapping rule*. Now the term *contextual* here aims to indicate that the instance and type mapping rules should also indicate the way in which those comparable properties can be accessed (in the respective information models) in order to be compared. This should be done with the help of relationship chains starting from the two instances/types

to be compared and ending in the properties that should be compared.
Figure 4.8 tries to explain it with the help of an example. A physical device, say X, has been represented in four different information models A, B, C and D. All rectangles in the figure represent instances of some types defined within the information model. In information model A, the digital representation of X is represented by DR-A, which is an instance of Type TR-A. In Information Model B, it has been represented as DR-B (instance of type TR-B) , in Information Model C it has been represented as DR-C (instance of type TR-C) and in Information Model D it has been represented as DR-D (instance of type TR-D). For the sake of simplicity, only the instances representing X have been shown with their respective types separated by a ':' sign, type definition on the left and the instance name on the right. The other instances shown in the figure also have some type definitions, however this has not been shown in the figure explicitly. Moreover, the relationships between the instances has been represented with the help of arrows and the names alongside the arrows are the names of those relationships.



IMR A-B:
DR-A -> HasProtocol -> CommProtocol -> HasNetworkAddress -> Network Address.Value = DR-B -> HasAddress -> Address.Value

IMR A-C:
DR-A -> HasProperty -> ManufacturerID.Value = DR-C-> ComponentOf -> Subsystem -> HasComponent -> ComponentIDs -> HasProperty -> MakersID.Value

IMR B-D:
DR-B -> HasAddress -> Address.Value = DR-D -> HasNetworkInfo -> NetworkInfo -> HasMACAddess -> MACAddress.Value

Figure 4.8: Instance and Type Mapping Rules

Since DR-A, DR-B, DR-C and DR-D represent the same physical entity X, therefore in the aggregated information model, there should exist a single instance representing X (as explained in 4.2.5). Now, the *IMR A-B* defines the condition which when satisfied would indicate that DR-A and DR-B are essentially two different representations of the same physical device. The rule indicates the properties to be compared and the relationship chain with the help of which that particular property should be accessed. This means that if the value of the property *Network Address* (in information model A) and that of the property *Address* (in Information Model B) are the same, then DR-A and DR-B essentially represent the same device. The access to this property has also been shown with the help of *relationship chains*. Similarly between information models A and C the *IMR A-C* suggests a comparison between the *Manufacturer ID* and the *MakersID* properties in models A and C respectively and between models B and D this is done by comparing the properties *Address* and *MACAddess* respectively (represented in the figure as *IMR B-D*) . It must be noted here that there is a transitive relationship between the IMR A-B and IMR B-D i.e essentially the same property in model B is being compared with properties in model A and model D in the two rules respectively. Therefore by transitivity, an instance mapping rule IMR A-D can be automatically generated. The auto-generation of rules will be discussed in Chapter 5 (5.2.1).

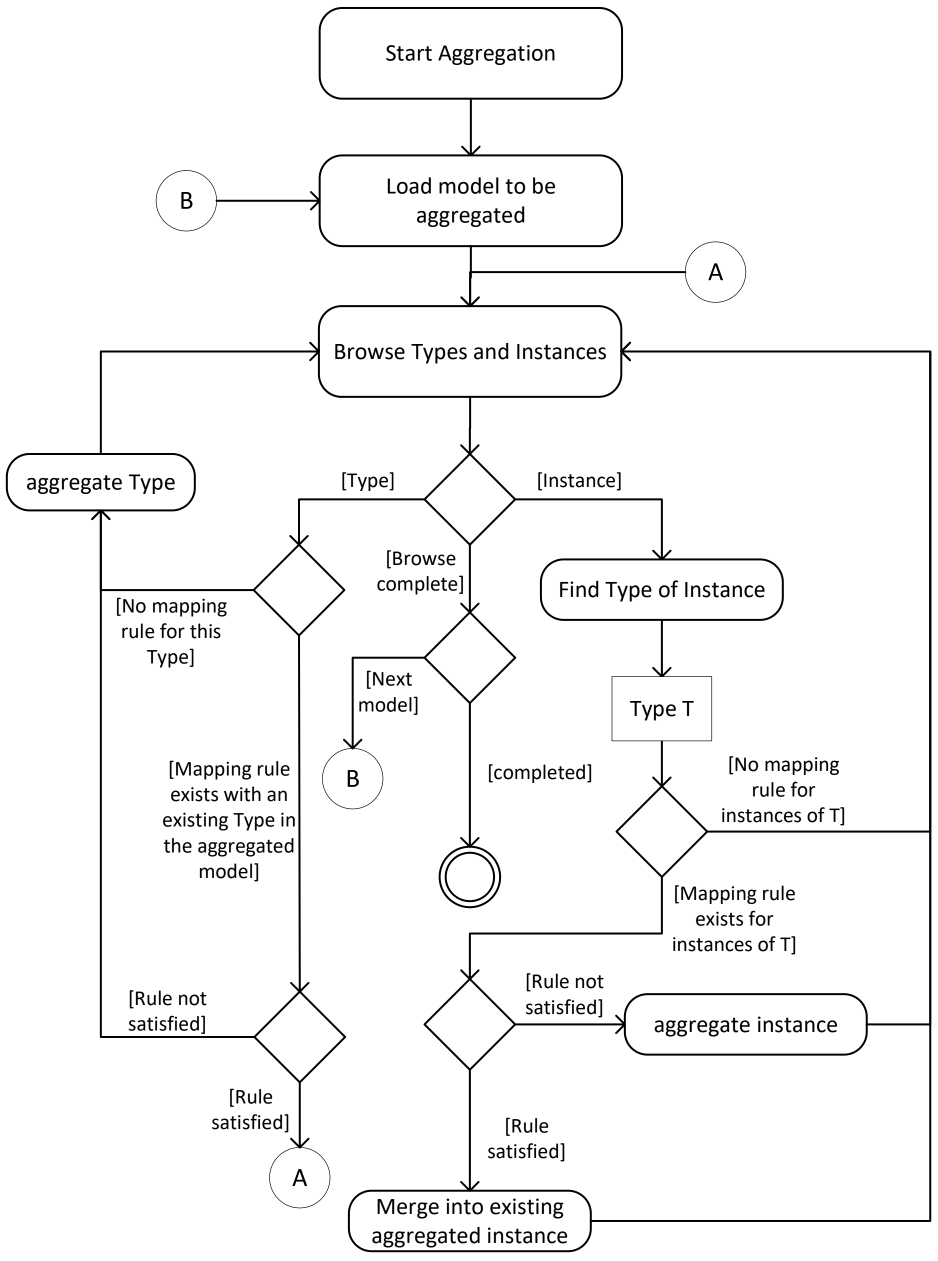

Start Aggregation
B
Load model to be aggregated
A
Browse Types and Instances
aggregate Type
[Type]
[Instance]
[Browse complete]
Find Type of Instance
[No mapping rule for this Type]
[Next model]
Type T
[Mapping rule exists with an existing Type in the aggregated model]
B
[completed]
[No mapping rule for instances of T]
[Mapping rule exists for instances of T]
[Rule not satisfied]
[Rule not satisfied]
aggregate instance
[Rule satisfied]
[Rule satisfied]
A
Merge into existing aggregated instance

Figure 4.9: Activity Diagram for the aggregation of types and instances

For better understanding, let us discuss the relationship chains shown in the figure (4.8) and how they should be interpreted and used. Mapping Rule A-B outlines when an instance in model A can be said to represent the same entity as that represented by an instance in model B. Let us assume that at the very beginning,the aggregation platform aggregates information model A. So after the first aggregation cycle, the aggregated information model will already contain the instances and types defined in information model A as at this point there are no pre-existing aggregated types or instances within the aggregated information model. After this, the aggregation platform starts to aggregate the instances defined in information model B one by one. For each instance in information model B, the instance mapping rule should be referred to, in order to see if a rule exists between this instance and any of the pre-existing instances in the aggregated model (in this case all instances aggregated from model A) . The first condition that should be compared, which has not been explicitly shown in the figure, is the comparison of type definitions of the instances. This is to say that looking at the mapping rule, it is clear that there is a rule between an instance of type TR-A and an instance of type TR-B. This means that while aggregating Model B, as per Mapping Rule A-B, only for the instances of type TR-B this rule should be considered for comparison of property values. Moreover only existing aggregated instances of type TR-A in the aggregated model should be compared with the aforementioned instances in model B. In order to compare the values of the parameters relationship chains should be followed to compare the values of the property instances e.g. Starting in order to compare an instance of TR-A and an instance of TR-B, starting at the instance of TR-A (in this case DR-A), the next hop instance via the relationship *HasProtocol* is the instance *CommProtocol.* From there following the relationship *HasNetworkAddress* we reach the instance *NetworkAddress.* Similarly in model B, starting at an instance of type TR-B (in this case DR-B), we can reach the instance *Address* by following the relationship *HasAddress.* The Mapping Rule A-B says that if the value of the two instances *NetworkAddress* and *Address* are the same, then it can be assured that both DR-A and DR-B are representations of the same entity, therefore the information obtained via the instance DR-B in model B should be merged within the single instance representing X in the aggregated model. How this merged information should be structured and visualised within the aggregated model and how the mapping rules should be represented are implementation specific details. An example implementation will be shown in the Realization and Review Chapter (Chapter 6).

4.2.7 Demonstration of a Merged Information Model with Graphs

This section demonstrates the concept of mapping rules introduced above and exemplifies how this concept can address the aggregation challenges introduced as part of the aggregation scenario mentioned in Section 4.2.4.

As mentioned in the previous chapter, Graph information models are versatile in the sense that nearly any information model, irrespective of the modelling approach used, can be translated into Graphs, thereby enabling fast information retrieval with Graph queries. Graph queries can also be used to create new graphs based on a certain matching criterion. Therefore, the aim of this demonstration is two-fold i.e.

- To investigate the criteria identifying the pump in the three information models (*Instance Mapping Rule*)
- To demonstrate an aggregated graph information model based on the above mapping rule

To achieve the first objective, three simple graph information models have been created for the pump (in the example scenario), each confirming to the corresponding standard information models i.e. DEXPI, PA-DIM and OPC 40223. The relationship between the resources in the graph have been represented following the standard RDF schema as the relationships described in the DEXPI, PA-DIM and OPC 40223 models were found to be directly mappable to the ones defined in the RDF schema. The resources have been represented as elliptical shapes. A resource representing a *Type* has been connected to the resources representing *instances* of that type, via *rdf:type* relationships. Please note that for the sake of simplicity, not all the properties (but a subset of the same) defined in the respective standards have been shown here. However, since not all three standards provide a standardised mapping to the semantic web[4], the respective relationships (edges) have been defined only for the current example. All of these properties can have values and if so, can be further connected to value nodes representing these values[5]. These value nodes have been depicted with rectangles in the figures. The following subsections briefly describe the example graph information models. Moreover, different namespaces have been used for resources that represent the instances than the namespaces for their respective types and standardised relationships[6], which is usually the case in a real modelling scenario as well.

DEXPI Graph Information Model

Figure 4.10 represents a graph information model that translates and uses the DEXPI information model. The figure shows only a subset of the entities (types,

[4]DEXPI does provide a standardised RDL and this has been considered in the Graph model for the DEXPI example

[5]for the sake of simplicity only the values of the nodes of interest have been depicted

[6]these namespaces are usually defined by the standard itself

instance declarations, properties and relationships) defined in the DEXPI model. In this model, the pump (mentioned in scenario 4.2.4) has been modelled as a resource (*eg1:PumpP1*) of *pca:CentrifugalPumpType* (defined by the DEXPI specification). In this example, *eg1* is chosen as the namespace containing the instantiations of the types defined in the *dexpi* and *pca* namespaces as defined by the DEXPI specification. As outlined by the *pca:CentrifugalPumpType*, the object *eg1:PumpP1* has several properties. This model shows the properties as *pca:DesignRotationalSpeed*, *pca: DesignVolumeFlowrate*, *dexpi:DesignPressureHead*, *pca:DifferentialPressure*, and *dexpi:TagNameAssignment*. In this case the property of interest is the *dexpi:Tag-NameAssignmentClass*. As defined by the specification, this property represents the tag number of the tagged item in the plant [Dex]. As can be inferred from the model, this tag number is a string literal "P1612-A". The RDF dataset for the graph can be referred to in the Listing 4.1.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dexpi: <http://sandbox.dexpi.org/rdl/> .
@prefix pca: <http://data.posccaesar.org/rdl/> .
@prefix eg1: <http://mygraphmodel.org/dexpiexample> .

eg1:PumpP1
rdf:type pca:CentrifugalPump;
pca:DesignRotationalSpeed  "42.24";
pca:DesignVolumeFlowrate "42.42";
dexpi:DesignPressureHead "42.4";
pca:DifferentialPressure "42";
dexpi:TagNameAssignmentClass "P1612-A".
```

Listing 4.1: RDF Dataset for the DEXPI example graph

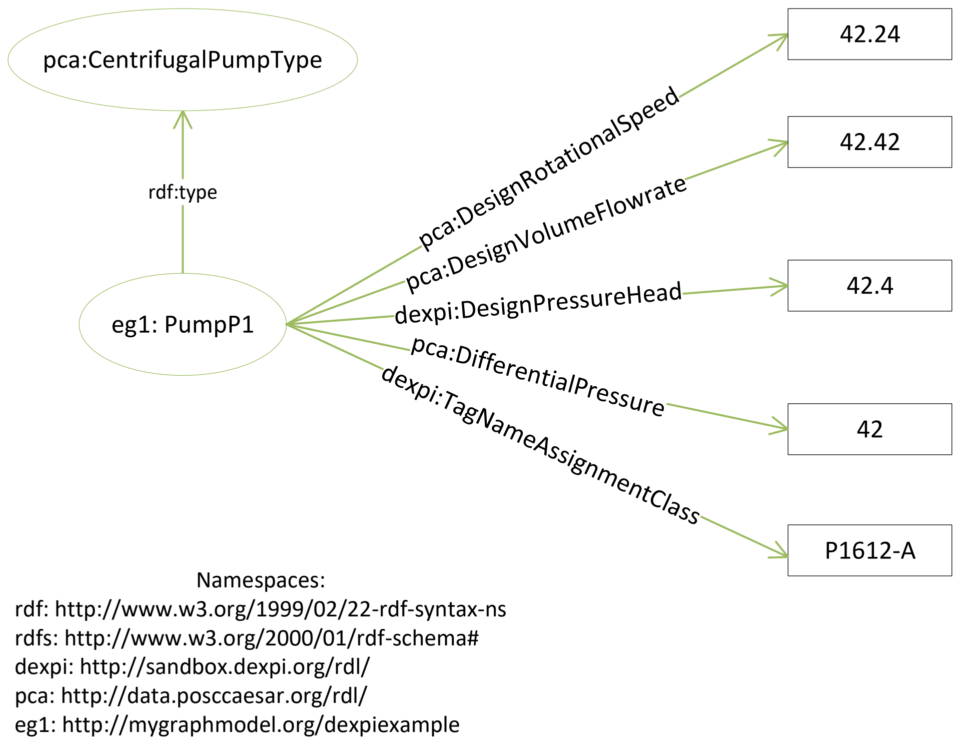

pca:CentrifugalPumpType
rdf:type
eg1: PumpP1
pca:DesignRotationalSpeed
pca:DesignVolumeFlowrate
dexpi:DesignPressureHead
pca:DifferentialPressure
dexpi:TagNameAssignmentClass
42.24
42.42
42.4
42
P1612-A
Namespaces:
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns
rdfs: http://www.w3.org/2000/01/rdf-schema#
dexpi: http://sandbox.dexpi.org/rdl/
pca: http://data.posccaesar.org/rdl/
eg1: http://mygraphmodel.org/dexpiexample

Figure 4.10: DEXPI Graph Information Model of the Pump

PA-DIM Graph Information Model

Figure 4.11 represents a graph information model that translates and uses the PA-DIM information model. The figure shows only a subset of the entities (types,instance declarations, properties and relationships) defined in the PA-DIM model. The namespaces (other than *rdf* and *rdfs*) used are *cdd* - for the IEC Common Data Dictionary [Cdd] referred to by the PA-DIM specification, *padim* - for the standardised type information defined in the PA-DIM specification and *eg2* - for the instances of the types defined in the PA-DIM specification (these namespaces are not standardised and have been used for a logical understanding of this example). In this model, the pump (mentioned in scenario 4.2.4) has been modelled as a resource (*eg2:PumpP2*) of *padim:PADIMType*. As outlined by the *PADIMType* (defined by the PA-DIM specification), the object *eg2:PumpP2* has several properties. This model shows the properties as *cdd:Manufacturer*, *cdd:SerialNumber*, *cdd:ProductInstanceUri*, *padim:DeviceHealth*, and *padim: SignalSet*. Furthermore the resource *eg2:SignalSet* contains (as a member) the resource *eg2:SignalS1* which is of the type *SignalType* (defined in the PA-DIM standard, but not shown in the figure). As defined by the *SignalType*, any instances of the same shall have a property *SignalTag*. This property has been represented by the resource *eg2:SignalTag* in the graph model and is of interest since it represents an alphanumeric sequence uniquely identifying a measuring or control point [Fou21a]. As can be inferred from the model, this tag number is a string literal "P1612-A". In this case there are three more properties (of the *eg2:PumpP2* resource) of interest namely the *cdd:Manufacturer*, *cdd:SerialNumber* and *cdd:ProductInstanceUri* which have values "XYZ Solutions GmbH","1234' and "XYZ-1234" respectively associated with them. The RDF dataset for the graph can be referred to in the Listing 4.2.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix cdd: <https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/> .
@prefix padim: <http://myexample.com/padim/rdl> .
@prefix eg2: <http://mygraphmodel.org/padimexample> .

eg2:PumpP2
rdf:type   padim:PADIMType ;
cdd:Manufacturer "XYZ␣Solutions␣GmbH";
cdd:SerialNumber "1234";
cdd:ProductInstanceUri "XYZ-1234";
padim:hasDeviceHealth padim:DeviceHealth ;
padim:hasSignalSet eg2:SignalSet .

eg2:SignalSet
padim:hasSignal eg2:SignalS1 .

eg2:SignalS1
cdd:SignalTag "P1612-A".
```

Listing 4.2: RDF Dataset for the PADIM example graph

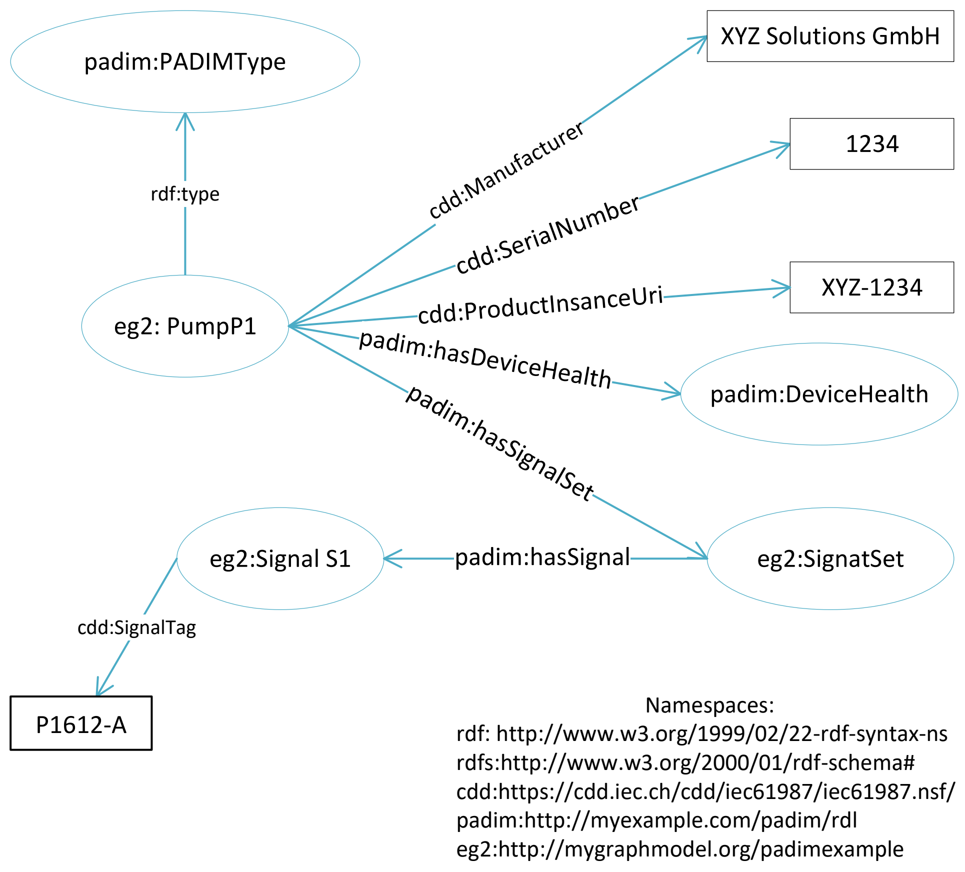


Figure 4.11: PA-DIM Graph Information Model of the Pump

OPC 40223 Graph Information Model

Figure 4.12 represents a graph information model that translates and uses the OPC 40223 information model. The namespaces (other than *rdf* and *rdfs*) used are *opc40223* - for the standardised type information defined in the OPC 40223 specification and *eg3* - for the instances of the types defined in the OPC 40223 specification (these namespaces are not standardised and have been used for a logical understanding of this example) [7]. The figure shows only a subset of the entities (types,instance declarations, properties and relationships) defined in the OPC 40223 model. In this model, the pump (mentioned in scenario 4.2.4) has been modelled as a resource (*eg3:PumpP3*) of *opc40223:PumpType*. As outlined

[7][Sch+19] introduces a standardised way to map OPC UA models to graph information models, but this example does not make use of the same for the sake of simplicity.

by the *PumpType* (defined by the OPC 40223 specification), its instance shall have several properties. This model shows these as resources *eg3:Identification, eg3:Maintenance, eg3:Operational and eg3:Ports*. The *eg3:Identification* resource further contains properties namely *opc40223:Manufacturer, opc40223:SerialNumber and opc40223: ProductInstanceUri*. In this case there are three properties of interest namely the *opc40223:Manufacturer*, the *opc40223:SerialNumber* and *opc40223: ProductInstanceUri* which have values “XYZ GmbH”, “SN#1234”, and “XYZ-1234” respectively associated with them.The RDF dataset for the graph can be referred to in the Listing 4.3.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix opc40223: <http://myexample.com/opc40223/rdl> .
@prefix eg3: <http://mygraphmodel.org/opc40223example>.

eg3:PumpP3
rdf:type opc40223:PumpType;
opc40223:hasIdentification eg3:Identification;
opc40223:hasMaintenance eg3:Maintenance;
opc40223:hasOperational eg3:Operational;
opc40223:hasPorts eg3:Ports.

eg3:Identification
opc40223:ProductInstanceUri "XYZ-1234";
opc40223:Manufacturer "XYZ␣GmbH";
opc40223:SerialNumber "SN#1234".
```

Listing 4.3: RDF Dataset for the OPC40223 example graph

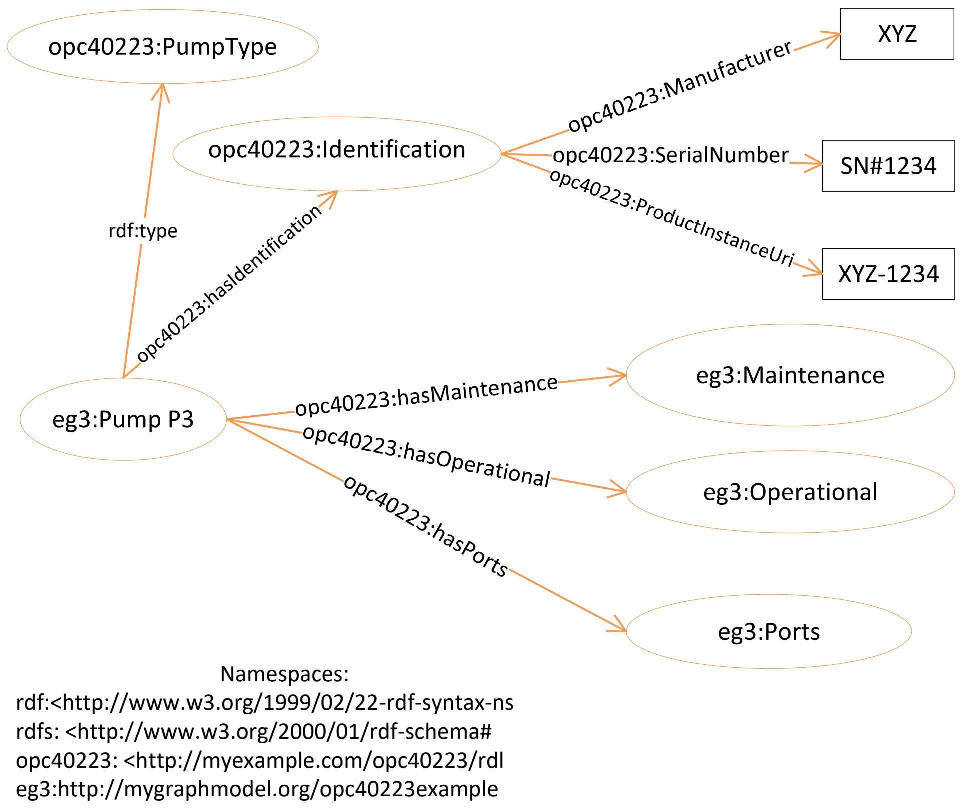


Figure 4.12: OPC 40223 Graph Information Model of the Pump

Instance Mapping Rule between DEXPI and PA-DIM models

As can be noticed from the figures 4.10 and 4.11, the assumption in case of this example is that the way to understand if both models are describing the same physical pump, would be to compare the values of the *TagNameAssignmentClass* property of the DEXPI model with that of the *SignalTag* property of the PA-DIM model. This assumption is based on the definitions of the two properties in the respective standards, treating the pump as a tagged item in the plant in case of its representation as per the DEXPI specification, and as a measuring or control point in case of its representation as per the PA-DIM specification. Thereby if the values of these propoerties match (as is the case in the example), it can be concluded that both resources *eg1:PumpP1* in the DEXPI Model and *eg2:PumpP2* in the PA-DIM model are representations of the same physical pump in two different information models. The listing 4.4 demonstrates a simple SPARQL query that can be used to match the instances of the pump between the aforementioned example RDF datasets for the DEXPI and PA-DIM based models shown in listings 4.1 and 4.2 respectively.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dexpi: <http://sandbox.dexpi.org/rdl/>
PREFIX pca: <http://data.posccaesar.org/rdl/>
PREFIX cdd: <https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/>
PREFIX padim: <http://myexample.com/padim/rdl>
PREFIX eg1: <http://mygraphmodel.org/dexpiexample>
PREFIX eg2: <http://mygraphmodel.org/padimexample>

SELECT ?pump1 ?pump2 ?productinstanceuri ?tag
WHERE {
        ?pump1 dexpi:TagNameAssignmentClass ?tag.
        ?pump2 cdd:ProductInstanceUri ?productinstanceuri.
        ?pump2 padim:hasSignalSet ?sigset.
        ?sigset padim:hasSignal ?sig.
        ?sig cdd:SignalTag ?tag.
}
```

Listing 4.4: SPARQL query to match the two pump instances between the DEXPI and PA-DIM models

Instance Mapping Rule between PA-DIM and OPC 40223 models

As can be noticed from the figures 4.11 and 4.12, the way to understand if both models are describing the same physical pump, would be to compare the value of the *ProductInstanceUri* property of the PA-DIM model with that of the *ProductInstanceUri* property of the OPC 40223 model. This assumption is based on the definitions of the two properties in the respective standards. If the respective values match (as is the case in the example), it can be concluded that both objects *eg2:PumpP2* in the PA-DIM Model and *eg3:PumpP3* in the OPC 40223 model are representations of the same physical pump in two different information models. Now, it is important to note here that although the definitions of the *Manufacturer* and *SerialNumber* properties are semantically similar, the standards do not define them as globally unique. Therefore, although these properties were chosen as properties of interest, they cannot be used as a criterion to match the instances representing the pump in the two models. The example also demontrates this fact where the values of the *Manufacturer* and *SerialNumber* properties in both models are indeed very similar but not exactly the same. In case of the *ProductInstanceUri* properties, they are defined (in both the standards) to be the globally unique identifier for the resource. The listing 4.4 demonstrates a simple SPARQL query that can be used to match the instances of the pump between the aforementioned example RDF datasets for the PA-DIM and OPC40223 based models shown in listings 4.2 and 4.3 respectively.

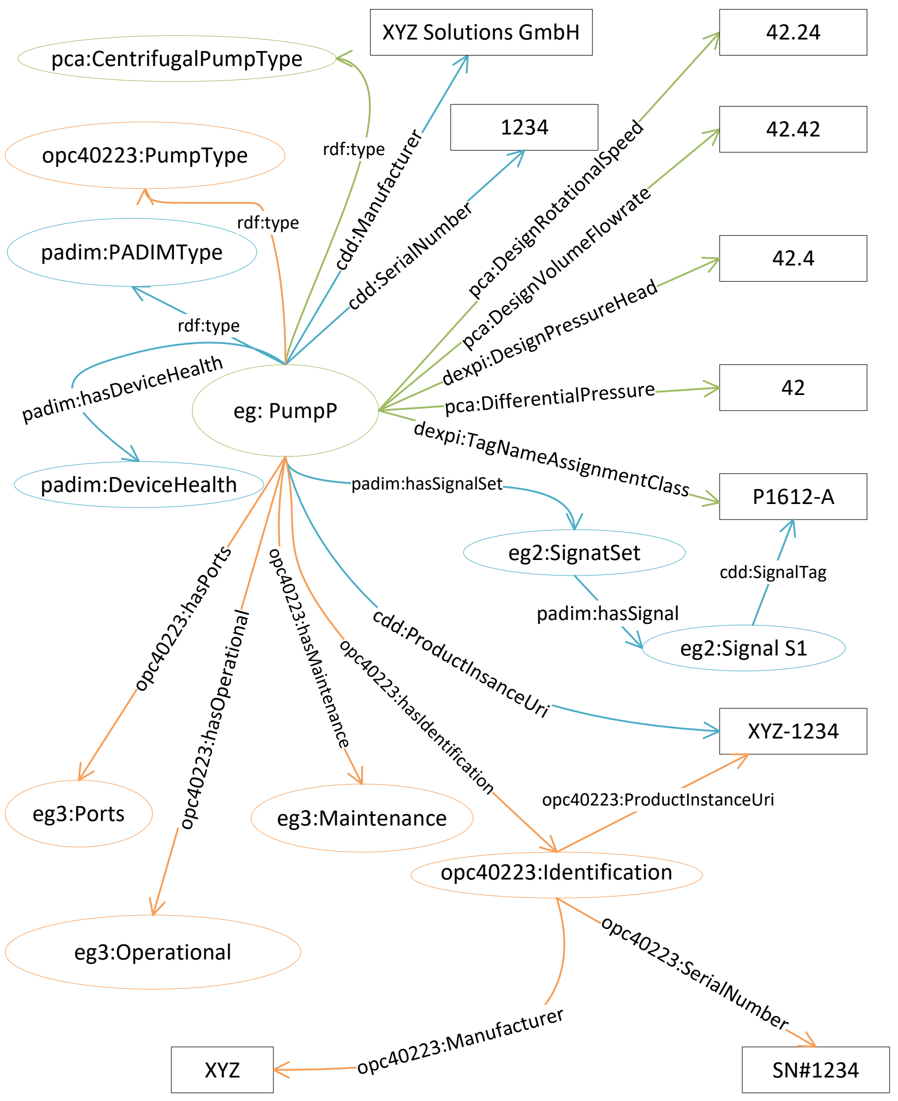
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX opc40223: <http://myexample.com/opc40223/rdl>
PREFIX cdd: <https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/>
PREFIX padim: <http://myexample.com/padim/rdl>
PREFIX eg2: <http://mygraphmodel.org/padimexample>
PREFIX eg3: <http://mygraphmodel.org/opc40223example>

SELECT ?pump2 ?pump3 ?piu2
WHERE {
        ?pump3 opc40223:hasIdentification ?ident3 .
        ?ident3 opc40223:ProductInstanceUri ?piu.
        ?pump2 cdd:ProductInstanceUri ?piu.
}
```

Listing 4.5: SPARQL query to match the two pump instances between the PA-DIM and OPC 40223 models

Merging the models using the Instance Mapping Rules

The three models shown in the figures 4.10, 4.11 and 4.12can be merged using the instance mapping rules discussed above. The merged model has been shown in Figure 4.13. As can be noticed from the figure, the three different objects representing the pump in three models have been merged to a single resource *eg:PumpP* in the merged model and all the relationships that were represented in the different models have been preserved in the merged model. The color coding used in Figure 4.10, Figure 4.11 and Figure 4.12 has been preserved in Figure 4.13 as well, so that it is easy to visualize which elements come from which graphs. Another thing to note in the merged graph is that the resource representing the merged instance of the pump (*eg:Pump*) has a *rdf:type* relationship with all the three types from the three graphs namely *pca:CentrifugalPumpType*, *opc40223:PumpType* and *padim:PADIMType*, which is theoretically possible from a graph modelling perspective but might not be possible in other modelling approaches e.g., in implementations which interpret this as multiple inheritance, this might not be allowed, in which case a merged resource representing the type might be considered. This merged graph expresses the non-redundancy requirement in the sense that the same value nodes are reachable via two different relationship paths (e.g. the value node *XYZ-1234* can be reached from the resource *eg:PumpP* either via its *opc40223:hasIdentification -¿ opc40223:Identification -¿ opc40223:ProductInstanceUri* path or via the *cdd:ProductInstanceUri* path). However the paths themselves are not considered to be redundant since an application that knows any one of these paths should be able to rely on the same in order to retrieve the value. The RDF dataset for the merged graph can be referred to in the Listing 4.6.



Namespaces:
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns
rdfs:http://www.w3.org/2000/01/rdf-schema#
cdd:https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/
padim:http://myexample.com/padim/rdl^
dexpi: http://sandbox.dexpi.org/rdl/
pca: http://data.posccaesar.org/rdl/
opc40223: <http://myexample.com/opc40223/rdl
eg1: http://mygraphmodel.org/dexpiexample
eg2:http://mygraphmodel.org/padimexample
eg3:http://mygraphmodel.org/opc40223example
eg:http://mygraphmodel.org/mergedgraphexample

Figure 4.13: Merged Graph Information Model of the Pump

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dexpi: <http://sandbox.dexpi.org/rdl/> .
@prefix pca: <http://data.posccaesar.org/rdl/> .
@prefix cdd: <https://cdd.iec.ch/cdd/iec61987/iec61987.nsf/> .
@prefix padim: <http://myexample.com/padim/rdl> .
@prefix opc40223: <http://myexample.com/opc40223/rdl> .
@prefix eg1: <http://mygraphmodel.org/dexpiexample> .
@prefix eg2: <http://mygraphmodel.org/padimexample> .
@prefix eg3: <http://mygraphmodel.org/opc40223example>.
@prefix eg: <http://mygraphmodel.org/mergedgraphexample>.

eg:PumpP
rdf:type pca:CentrifugalPump;
rdf:type   padim:PADIMType ;
rdf:type opc40223:PumpType;
pca:DesignRotationalSpeed  "42.24";
pca:DesignVolumeFlowrate "42.42";
dexpi:DesignPressureHead "42.4";
pca:DifferentialPressure "42";
dexpi:TagNameAssignmentClass "P1612-A";
cdd:Manufacturer "XYZ␣Solutions␣GmbH";
cdd:SerialNumber "1234";
cdd:ProductInstanceUri "XYZ-1234";
padim:hasDeviceHealth padim:DeviceHealth ;
padim:hasSignalSet eg2:SignalSet ;
opc40223:hasIdentification eg3:Identification;
opc40223:hasMaintenance eg3:Maintenance;
opc40223:hasOperational eg3:Operational;
opc40223:hasPorts eg3:Ports .

eg2:SignalSet
padim:hasSignal eg2:SignalS1 .

eg2:SignalS1
cdd:SignalTag "P1612-A".

eg3:Identification
opc40223:ProductInstanceUri "XYZ-1234";
opc40223:Manufacturer "XYZ␣GmbH";
opc40223:SerialNumber "SN#1234".
```

Listing 4.6: RDF Dataset for the Merged graph

4.3 Base Model of Information Aggregation

The objective of this section is to use the results of Information Models-Analysis, to abstract and to structure them in the form of a Base Model for Information Aggregation. Therefore, the following sections introduce different levels of the Base Model which structure the different Information-Model Management relevant artifacts in a functional hierarchy. In a functional hierarchy the upper layers take information from the lower layers for further processing and representation. In such a structure, there is a clear separation of responsibilities of the different layers and how they handle data and information. The Base Model has been divided into 3 main layers (Figure 4.14) namely, the *Aggregation Communication Model* (comprising of the *Communication Sequences* and the *Addressing Information* sub-modules), the *Aggregated Information Model* (comprising of the *Basic Methods* and the *Data and State Model* sub-modules) and the *Consumer Operation Model* (comprising of the *Graphical User Interface* and the *Consumer Business Logic* sub-modules). The following sub-sections will describe each of the layers separately.

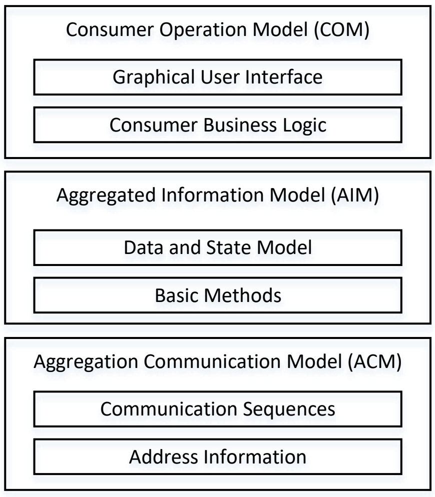


Figure 4.14: The Base Model

4.3.1 Aggregated Information Model

The AIM is the core of the Base Model. It contains all the information of all the UIMs and makes them available to the upper layer in the Base Model, i.e to the *Consumer Operation Model* or any other consumer applications that might need the information. The access to the individual UIMs is carried out via the AIM. The AIM, on the other hand communicates with the information sources with the help of the Aggregation Communication Model (Section 4.3.2), which contains the required information in order to establish communication with the underlying sources of information. The AIM is further divided into two sub-modules namely the *Data and State Model* and the *Basic Methods*. The following subsections describe the respective sub-modules.

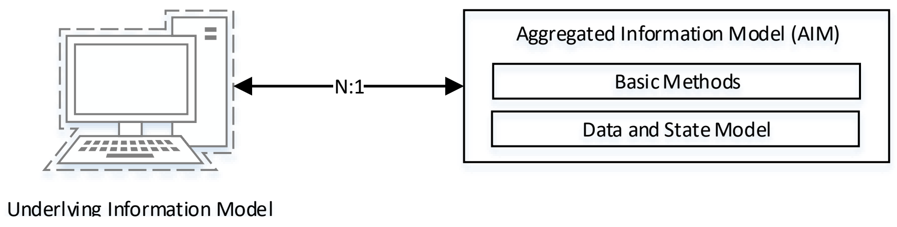

Aggregated Information Model (AIM)
Basic Methods
Data and State Model
N:1
Underlying Information Model

Figure 4.15: Aggregated Information Model

Data and State Model

The *Data and State Model (DSM)* contains all the data which the UIMs (which are being aggregated) have made available to its consumers. The consistency of the data is safeguarded by the DSM i.e. the *State* of the data in the AIM and UIMs should be consistent at all times. The DSM layer, with the help of the ACM, reads the data values from the UIM and can write it back to the source after manipulation of the same. The data values can be simple variable or parameter values, or values complying to complex data structures. In addition to the data values, the DSM also contains metadata about the same e.g. Name, Data Type, Scalar/Vector and also information about other dependencies (e.g. Alarms and Warning threshold etc.).

Definition 4.3 ***Data and State Model***
The DSM defines the data elements that the AIM exposes to its consumers. It maintains the consistency of the data by using the dynamic and logical dependencies between them

As described in the definition, the DSM contains all the logical and dynamic dependencies of the data elements in order to safeguard the consistency of the same. This entails the access rights to the concerned data value. Data consistency is specifically important in cases e.g. where there is a dependency between a data value and its corresponding alarm threshold condition as the breach of this threshold should trigger an alarm. Therefore it is not only important to maintain the consistency of the data but also the corresponding threshold limits set for the particular data value in the underlying model. Therefore it is the responsibility of the DSM to observe such conditions and maintain the consistency of the data. Even in case of a data value subscription, the DSM is responsible to provide consistent notifications from the underlying systems about the data values being subscribed to, by consumer applications. Moreover, it could also be possible that in order to always provide a consistent view of the data from the underlying system, there would be a need to get the data from the underlying system following a particular protocol or communication sequence. This functionality would be the responsibility of the Aggregation Communication Model (Section 4.3.2).

Basic Methods

The *Basic Methods (BM)* consist of the atomic functions that can be carried out on the data elements belonging to the underlying systems that are aggregated e.g. *Read* to read a data value from an underlying source and *Write* to write a value back to the underlying data element. These functions should directly be made available by the underlying system, which can in-turn be called from the aggregation layer as and when required. In other words, a *Read* method called to read a data value of the aggregated model should trigger the *Read* method of the underlying system to read the corresponding data value in the underlying system.

Definition 4.4 ***Basic Methods***
The BM layer handles the basic atomic methods inherently provided by the UIMs. These methods which are provided by the underlying system in order to access its data shall be outsourced with the help of the DSM of the aggregation layer to provide indirect access (via the AIM) to the data of the underlying system.

4.3.2 Aggregation Communication Model

The *Aggregation Communication Model (ACM)* manages the communication with the underlying system. Therefore the ACM contains all the parameters required to establish the communication e.g. addressing information of the underlying system and the communication sequences that it needs to follow to access the information from the UIM. Therefore within the scope of this thesis report, the ACM deals with those communication related activities that are relevant in order to access the UIM and its functions. The ACM encapsulates protocol specific communication. In general, the UIMs can specify different communication protocols in order to establish communication with them and access their data and functions. In such a case, each such communication scenario would be handled by its own ACM. So in the event of aggregation of multiple information models, with different communication protocols, there will be a different ACM with respect to each protocol, however all these models would still provide their information to the DSM (in the AIM). The *Addressing Information* and the *Communication Sequences* sub-modules together build up the ACM. In conclusion, it can be said that the relationship between the AIM and the ACM is that the ACM contains technology specific information about *how* the data-exchange will be performed between the AIM and the UIMs.

Definition 4.5 ***Aggregation Communication Model***
The ACM handles the UIM-specific aspects of the communication. It encapsulates the protocol specific addressing mechanism as well as the protocol-specific communication mechanisms. For each UIM (being aggregated) there exists exactly one ACM (1:1 relationship).

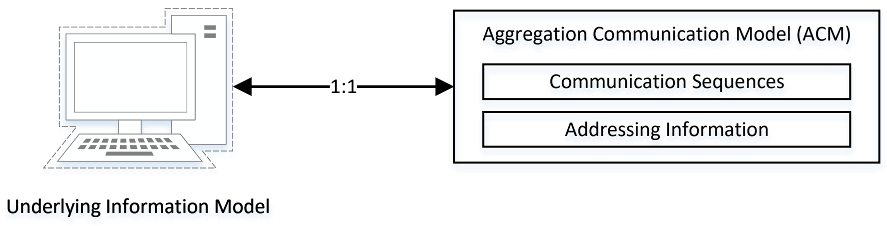

Aggregation Communication Model (ACM)
Communication Sequences
Addressing Information
1:1
Underlying Information Model

Figure 4.16: Aggregation Communication Model

Addressing Information

The *Addressing Information (AI)* layer depicts the UIM-specific internal addressing of the data elements and functions, as well as the protocol-specific addressing mechanism of the underlying system itself, which are mandatory to establish communication with the underlying system and then to access the data and functions modeled by the UIM.

Definition 4.6 ***Addressing Information***
AI contains the protocol specific addressing information (to connect to the underlying system) and the information model specific addressing information to locate and access the data elements and functions within the information model).The ACM accesses the data values and the functions using the addressing information contained by the AI.

Communication Sequences

The *Communication Sequences (CS)* deals with the sequence in which the communication between the consumer applications and the underlying systems need to occur, when the aggregation layer serves the data and the functions represented by the UIMs. For example when a consumer application wants to read a data value, it sends the request to the aggregation layer, the aggregation layer should then retrieve the information from the respective underlying system and serve the request back to the consumer. In the case of an event or an alarm, the sequence will be in the reverse order i.e. the event should be propagated from the underlying system, via the aggregation layer, to the respective consumer interested in the event.

Definition 4.7 ***Communication Sequences***
CS represents the path that any communication between the consumers and the sources of information takes in order to exchange data and information. This includes the sequence in which functions and events are handled by the ACM.

4.3.3 Consumer Operation Model

The core of the *Consumer Operation Model (COM)* is formed by the Use-specific operating user interface where the user sees the information modeled in the AIM. The AIM will provide an aggregated view of several information models in one place, however the COM will selectively pick only the relevant data and functionalities which are relevant to its particular use case, so the COM is more focused on a specific objective. For example, it is possible for multiple COMs to concentrate on different subsets of the AIM in order to analyze the data and information for different purposes. This suggests an n:1 relationship between the COM and the AIM. Thus the COM can be further divided into two sub-modules viz. the *Consumer Business Logic*, which simplifies the use-case specific operation criteria, and the *Graphical User Interface* which displays the data and functionalities depending upon the use-case. The following sections will further discuss the *Consumer Business Logic* and the *Graphical User Interface* layers.

Definition 4.8 ***Consumer Operation Model***
A COM contains the use-case specific operation interface which can used to interact with the data and functions from the AIM depending upon a particular use-case. The COM provides access to the data and functions of the UIMs via the AIM.

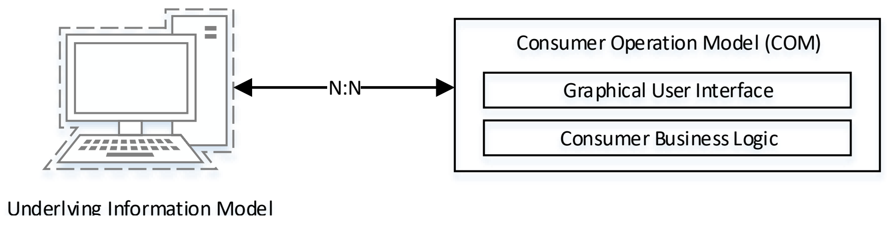


Figure 4.17: Consumer Operation Model

Consumer Business Logic

The *Consumer Business Logic (CBL)* layer contains all the high level logic elements relevant in order to support use-case specific operations on the aggregated data and functions made available by the AIM. It makes complex processing functions available to the *Graphical User Interface.* In other words, a CBL function can be designed to collect data values from different UIMs (via the AIM), process them and forward the result to the *Graphical User Interface* to be displayed to the user. For example, let us consider a CBL designed to analyze the energy consumption of a system with respect to the topological arrangement of the components in a system. In such a case, the CBL can get the aggregated information (from the AIM) about the topology and energy consumption of each component, process it according to its well-defined business logic and then formulate the result of the analysis.
In the reverse direction, the CBL can also be designed to carry out performance/-efficiency related analytics based on the data from the underlying systems, and can set parameters in the underlying system via the aggregation layer. This can be represented in the form of recommended settings for performance improvement and visualized in the *Graphical User Interface.* For example in the same scenario (discussed above), after calculating the energy efficiency (based on energy consumption measurements), the CBL can develop functions to suggest parameter set points for energy optimization. These set points can then be sent by the user to the underlying systems with the help of the *Graphical User Interface.*

Definition 4.9 ***Consumer Business Logic***
The CBL layer provides additional functional logic which work on the data from the underlying systems.It makes the usage of these functions available to the user from the Graphical User Interface. These use-case specific functions can collect data selectively from the underlying systems, manipulate them according to a pre-defined algorithm and then send data set points to the underlying systems.

Graphical User Interface

The *Graphical User Interface (GUI)* is the system boundary over which the operator can interact with the aggregated system. The GUI represents the use-case specific data elements functionalities available from the underlying systems and additional functions to manipulate the data in the underlying system. The functions are implemented as per the CBL.

Definition 4.10 ***Graphical User Interface***
The GUI provides a visual representation of the data elements and functionalities inherently provided by the underlying systems. It also provides the capability to manipulate the data elements with the help of algorithms developed according to the CBL.

4.4 Meta-model of Information Aggregation

The objective of this section is to develop a meta-model for information aggregation which can be used to develop a well-defined architecture for the solution introduced in this chapter. The meta-model is to be derived from the abstract Base Model (Section 4.3) consisting of the AIM (Aggregated Information Model), ACM (Aggregation Communication Model) and the COM (Consumer Operation Model). The methodology adopted for the development of the meta model is according to the Meta Object Facility (MOF) introduced by the Object Management Group (OMG). The MOF defines four model levels (Figure 4.18). The top level M3 is the Meta-Meta-Model which is the language used in the MOF to define the meta-models at the M2 level. The specification of *Unified Modelling Language (UML)* is an example of a M2 Level model, i.e. the model that is used to describe UML itself. M2 models are used to describe the elements in the M1 layer. Therefore M1 itself consists of models written in UML. The layer M0 is the data layer and models the concrete real world entities based on the models defined on the M1 layer.

As per the above discussion, with respect to this thesis, the meta-model for Information Aggregation will be a refined version of the Base Model (Level M2). This meta-model will be used to define concrete models (Level M1) for the types of Information Models (which will be aggregated). The instances (platforms exposing the information models) of the concrete Information Models will lie at Level M0. Thus, the meta-model represents the basis for the representation of concrete UIM types. It defines the building blocks depending upon which the Information Model-Management relevant aspects of an UIM are identified. UML has been used as the modeling language of choice to model the meta-model. It is a well-known and

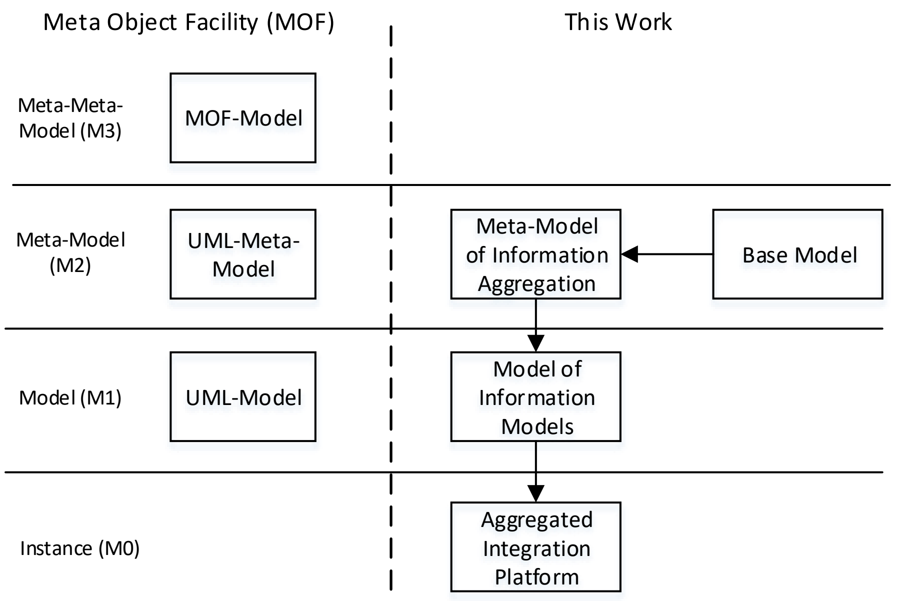

Meta Object Facility (MOF)
This Work
Meta-Meta-Model (M3)
MOF-Model
Meta-Model (M2)
UML-Meta-Model
Meta-Model of Information Aggregation
Base Model
Model (M1)
UML-Model
Model of Information Models
Instance (M0)
Aggregated Integration Platform

Figure 4.18: Modeling Levels of MOF [OMG06]

widely accepted language for graphical modeling. The following sections aim to build the meta-model and discuss the building blocks of the model.

4.4.1 Basic Structure of the Meta-model

Figure 4.19 shows the basic structure of the meta-model which depicts the Base Model presented in Section 4.3. A UIM serves as the basic element for this thesis. The Information-Model Management related activities are performed on the UIM by the aggregation layer with the help of the AIM (Aggregated Information Model), which aggregates all the data (and its dependencies) represented in the UIM. The AIM in turn contains a DSM (Data and State Model) and BM (Basic Methods) sub-layers, both of which will be further refined in the forthcoming sub-sections.

To handle the communication with the UIMs, the meta-model consists of ACMs. Each UIM has exactly one corresponding ACM, which in turn consists of the levels AI (Addressing Information) and CS (Communication Sequences). Depending upon the use-case for the usage of the aggregated model, multiple COMs (Consumer Operation Model) can co-exist. Each COM consists of a GUI (Graphical User Interface) and possibly a CBL (Consumer Business Logic) sub-layer. The GUI and the CBL sub-layers will be further refined. The Information

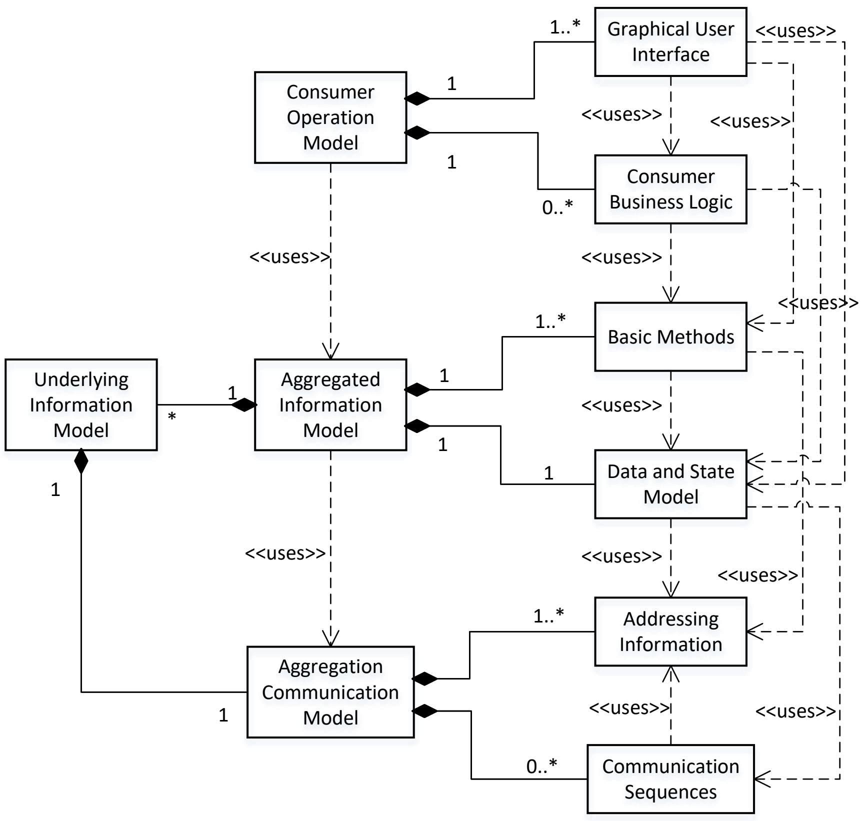


Figure 4.19: Basic Structure of the Meta-model

Model-Management related activities are then carried out by an operator via the GUI which acts the interface between the operator and the aggregated system. The GUI can visualize the elements modeled by the UIM with the help of the DSM (of the AIM). In order to use the data and information of the UIM, the GUI makes use of the BM sub-layer. The BM layer works in conjugation with the DSM layer in order to carry out operations on the data-elements modeled by the UIM. If higher-level analysis of the aggregated model is the requirement for a certain use-case, this would be implemented as a function as defined by the CBL layer and would be available for use via the GUI. Therefore the CBL-layer uses the DSM and the BM layers too. The following sections discuss each of the layers and presents the class diagrams for each of them. Only the classes directly within that layer are depicted with their respective members. The classes from other layers can be seen as placeholders which have been detailed in their appropriate sections.

4.4.2 Data and State Model

The DSM forms the basis for the functioning of the subsequent BM, CBL and GUI layers. The DSM essentially consists of the elements that represent the ones that constitute the UIMs. There are two basic elements in the DSM of the AIM namely the *Instance Element*, the *Type Element.* These are the basic building blocks of an information model as mentioned in 4.2.5. This means that each of these elements can be extended and further specified to form more elements in the information model. For example the *Type Element* represents a collection of all kinds of *types* that can be instantiated *viz. Object Types*, *Data Types*, *Relationship Types*, which can again be further extended and specialized. Both the basic elements can have associations with each other. These associations are called *Relationships* and are represented by *Relationship Elements* which is a subset of *Instance Elements.* A *Relationship Element* is an instantiation of a *Relationship Type.* In other words any element in an information model can have a association with any other element and this association is known as the *relationship* between the two elements. The *Instance Element* has another association with itself. In some cases, this association represents the *Instance Mapping Rule* of this instance element with another instance element in another information model. Similarly a *Type Element* in an aggregation scenario also has an association with other *Type Elements.* In case of same/similar types from different UIMs this association represents the *Type Mapping Rule* which maps this *Type Element* to the same/similar *Type Element* in another information model.
Both the *Instance Element* and the *Type Element* have the *Name, Description, ID, Type, Access* attributes which contain the meta-information about the elements. Figure 4.20 shows the UML representation of these elements.

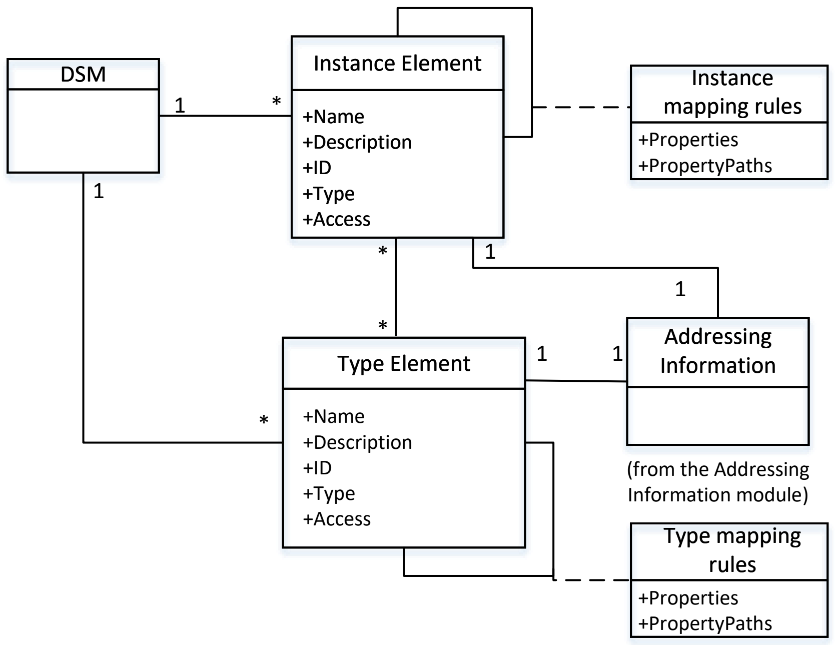


Figure 4.20: Data and State Model

4.4.3 Basic Methods

The BM level can contain multiple functions. These functions are used to interact with the UIM. For a particular UIM, the availability of these functions depend on the roles/rights of the consumer/system accessing the UIM. One such function will always have a name (*Name*), a description (*Info*), an availability (*Validity*) as well as a working logic (*Algorithm*). The input parameters needed in order to call the method are represented by the class *InputArgument*. The input parameters have a *Name*, a *Value* and data type *Type*. A method can give back some results after the execution of the working logic on the input parameters (if any), these results are represented by a *Return Value* class. Each result will in turn have a *Name*, a *Value* and a data type *Type*. The execution of the methods depends on whether the information consumer has the appropriate rights to call the method. This dependence of the method's availability is represented by the *Access* class. In order to explore the UIMs, the BM layer needs to know the address information about the UIM itself as well as its elements, in order to connect to it and call the available methods. Since BM methods access the data and information of the UIM, there is a direct association of BM with the DSM.

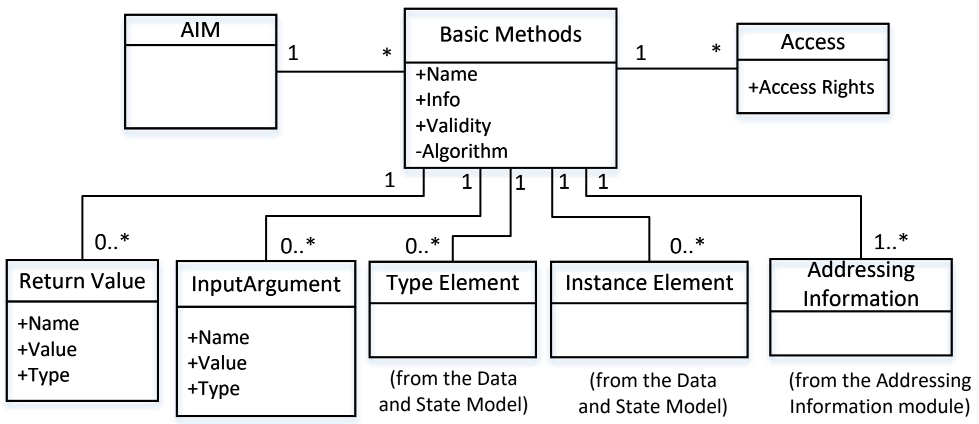


Figure 4.21: Basic Aggregation Methods

4.4.4 Addressing Information

The UIM can be hosted by different kinds of platforms following their own communication protocol. Thus, the addressing information contains the address of the UIM itself (how to access the UIM) and the UIM elements (how to access the elements within the UIM). This can depend on the particular communication protocol and how the UIM element can be reached according the modeling of the UIM. A UIM could support multiple communication protocols, therefore it can be reachable at multiple protocol specific addresses. The data and functions of a UIM will have an address within the UIM, so the the methods from the BM can interact with them.

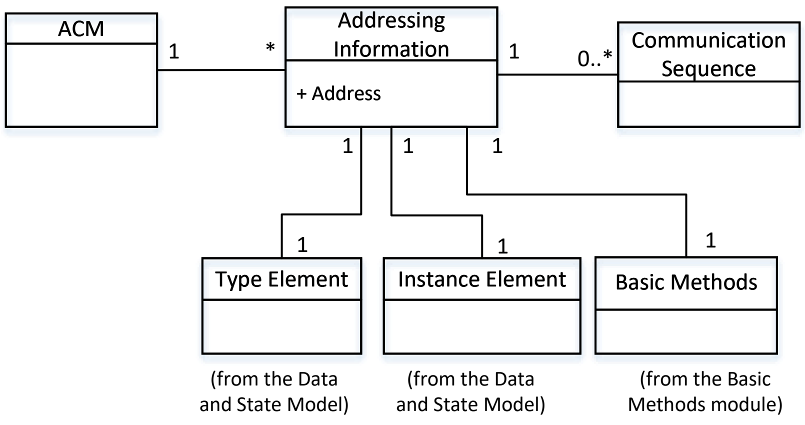


Figure 4.22: Addressing Information

4.4.5 Communication Sequences

The communication sequences define the logical flow of information between the UIMs and the consumers. Some information models do not provide direct read or write access to their data elements. In such a case, the *Communication Sequence* defines the UIM specific workflow to be followed in order to gain access to the data elements. The *Communication Sequence* contains a working logic (*Algorithm*) detailing the required workflow to be followed to be able to access the data elements in that model. This can also be dependent on the communication protocol used to communicate with the UIM. This layer can also be left empty if there are no specific communication sequences to be followed.

4.4.6 Consumer Business Logic

The CBL layer contains use-case specific operating functions. In the case when a COM doesn't have a specific use-case for the aggregated information in the AIM, this layer can also be left empty. Similar to the BM, the CBL consists of a name (*Name*), a possible result which could be returned (*Return Value*) and possible input parameters (*InputArgument*). A CBL is also associated with roles which define the availability of CBL functions depending upon access rights of the consumers.

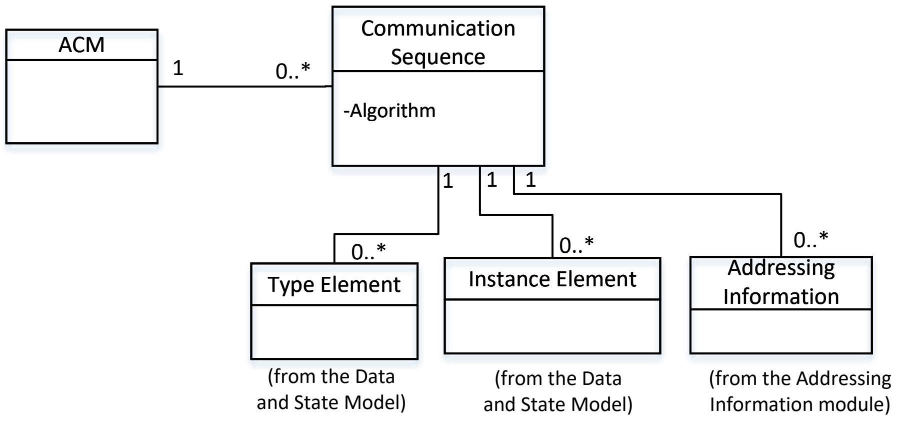


Figure 4.23: Communication Sequences

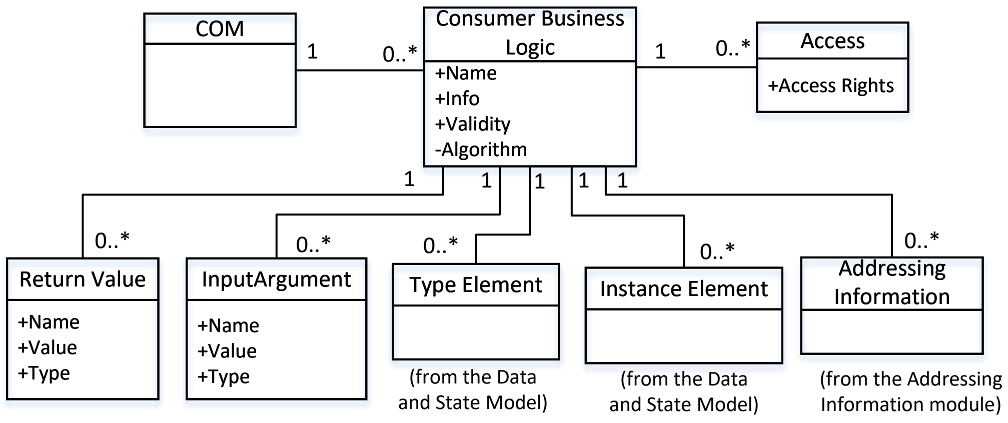


Figure 4.24: Consumer Business Logic

4.4.7 Graphical User Interface

The GUI represents the interface between the user and the system. The basic element *Consumer Operation* represents the actual operation to be carried out on the AIM. The Consumer Operation can be associated with one or more *Access* instances, which can control the access rights to the AIM. It provides a visual component which visualizes the information collected from the AIM and also provides visual elements to run the functions provided by the UIM and the use-case specific functionalities defined by the CBL-layer. These have been represented by the *Visual Element* class in the Figure 4.25. The Visual-Elements are therefore directly connected to the data elements in the DSM and to the functional elements in the BM levels. The types and scope of these Visual-Elements are defined and limited by the UIM and the user-roles.

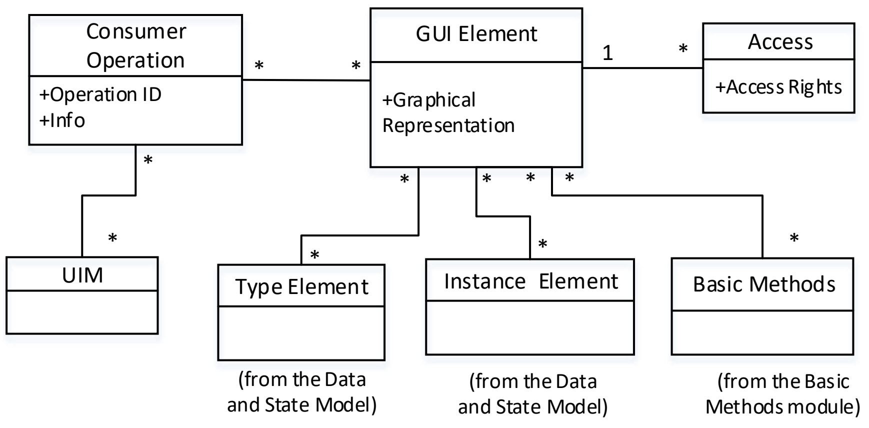

Consumer Operation
+Operation ID
+Info
GUI Element
+Graphical Representation
Access
+Access Rights
UIM
Type Element
Instance Element
Basic Methods
(from the Data and State Model)
(from the Data and State Model)
(from the Basic Methods module)
1
*

Figure 4.25: Graphical User Interface

4.5 Summary

In this chapter, the meta-model, which acts as the basis for the Aggregation of Information Models, was developed and discussed. Section 4.2 analyses the actors and the use-cases which deal with the Information Model-Management. From the discussed use-cases the functional and non-functional requirements were outlined in section 4.2.2. This was followed by an analysis of the available information models in the automation industry and the building blocks of a typical information model were identified in section 4.2.3. Further considerations about the aggregation of information models were observed and the challenge to merge the modeled types and instances was identified in the following subsections. In section 4.3 the base model for the aggregation of models was developed, which was then further refined to ultimately develop the meta-model in section 4.4. The main challenge for this chapter was to design a model which can be used to design a complete architecture for the development of a solution to aggregate multiple information models in a unified manner on a single platform. Such an architecture will be formulated in Chapter 5 of this thesis. Chapter 5 will address the basic concept for the solution development, introduced in section 4.1 and refine it further to develop and Architecture based on the Meta-model introduced in this chapter.

Chapter 5

Architecture of the Aggregated Integration Concept

Contents

5.1 Architecture Concept 110

5.1.1 Overview of the Architecture Concept 110

5.1.2 Procedure for the Architecture development 113

5.2 Structural Model 114

5.2.1 Conceptual Class Diagrams of the platform modules . . 114

5.3 Definition of Interfaces 122

5.3.1 Login-Interface 124

5.3.2 AIM-Interface 124

5.3.3 COM-Interface 125

5.3.4 Mapping-Interface 125

5.3.5 ACM-Interface 126

5.3.6 Configuration-Interface 126

5.4 Behavioral Model 127

5.4.1 Read Access to a UIM element 127

5.4.2 Write Access to a UIM element 129

5.4.3 Subscribing to a UIM instance 130

5.4.4 Addition of a UIM element by Consumer 132

5.4.5 Deletion of a UIM element by Consumer 133

5.4.6 Dynamic Addition of a UIM element 135

5.4.7 Dynamic Deletion of a UIM element 136
5.5 Summary 137

5.1 Architecture Concept

This section provides an overview of the basic concept architecture of the aggregated integration platform. Section 5.1.1 describes the structure of the components of the meta-model (described in Chapter 4) namely the AIM (Aggregated Information Model), ACM (Aggregation Communication Model) and the COM (Consumer Operation Model), within an architectural concept. In the next section (section 5.1.2), the concept architecture is refined to define a software-architecture of the Aggregated Integration Platform.

5.1.1 Overview of the Architecture Concept

Based on the solution concept for the aggregated integration platform introduced in Chapter 4, a three layer Meta-model consisting of the COM, AIM and the ACM, was developed. The AIM and the ACM are the UIM specific layers which deal with the representation of the UIM and entail its functionality in the aggregated environment. The COM layer is however use-case specific and enables the consumer to use the information from the AIM as defined by the particular use-case in consideration. This clear distinction between the UIM dependent (AIM and ACM) and use-case specific (COM) layers, forms the basis for the Architecture concept. This proves to be a good foundation for the usage of the Producer-Consumer design pattern[1] for this thesis [Bus]. In such an architecture a *Producer* offers certain services, which can be consumed by one or more *Consumers.* In this thesis, the various use-case specific COMs need to utilize the information provided by the single AIM. Therefore in light of the meta-model, it can be said that the Producer offers services, using which the Consumers can access the data and information of the UIMs. This Producer will be denoted as the *Aggregated Information Provider* (AIP) in 5.1 and in the forthcoming sections. In order to communicate with the UIMs, the AIP will make use of the information provided by the ACM. The Consumers are designed according to their respective COMs, which define the use-case specific requirements for information consumption. The GUI component of the consumers then represent the user-interface of the particular COM. The Consumer that do not require a GUI component, can directly access the data

[1]Design patterns in software development define proven basic methodologies used to develop a software system.The design pattern is chosen depending upon the requirements of such a software keeping the boundary conditions in mind.

and information made available by the AIM. These Consumers will be henceforth called as *Aggregated Information Consumer*(AIC).The AIP in itself can be said to contain a collection of consumers, each of which consume the information provided by the underlying producer (UIM). Therefore the AIP acts as a consumer for the UIM and as a producer (although it is technically not producing the information, rather providing access to it) for the AICs. The AIM contains the *Instance and Type Mapping rules* which should be accessible to the AIP and used by it during the aggregation of a UIM [2]. These mapping rules could be local or global (depending upon the type of deployment) and should be saved in a repository accessible to the AIP before it starts to aggregate a new UIM. Thus this thesis proposes the use of a *Local Repository* and/or a *Global Repository* where the instance and type mapping rules and be saved and accessed by different AIPs as per the deployment scope. This deployment scenario will be discussed in a greater detail in the forthcoming Section 5.2.1. It must be noted here that all security related aspects (secure communication of information and user data management) have not been considered in this architecture. All security related aspects must be considered while implementing this architecture.

[2]These mapping rules should be provided by the UIM designers who work in collaboration with each other to identify the rules that define when and how an instance/type should be mapped to separate or merged instances/types in the AIM (see 4.2.6)

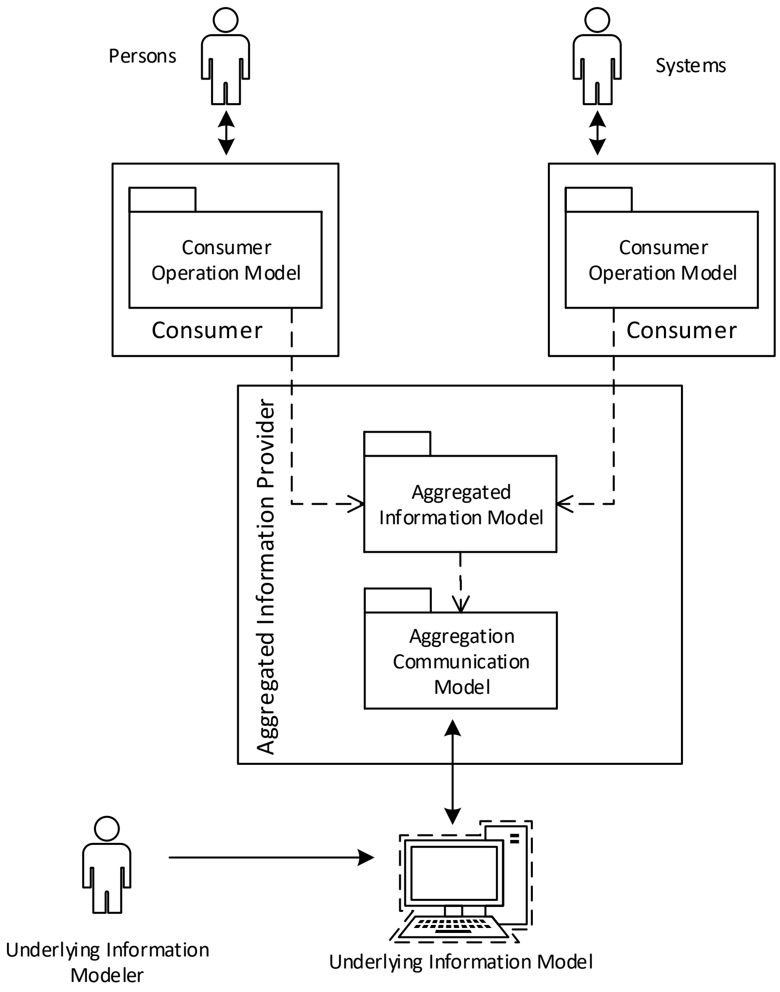

Persons
Systems
Consumer Operation Model
Consumer
Consumer Operation Model
Consumer
Aggregated Information Provider
Aggregated Information Model
Aggregation Communication Model
Underlying Information Modeler
Underlying Information Model

Figure 5.1: The Architecture Concept

So as described, the *Mapping Rules* will be used as a required unit for the aggregation of information models and needs to be made accessible to the AIP. The AIP however consists of the AIM and the ACM only. The *Mapping Rules* could also be proposed to be made a part of the UIM, however this brings additional challenges of defining how many mapping rules should an information model have with respect to how many other information models? If such a proposal is made, it would also mean an additional change to be made in all existing information models, which is a massive effort in itself. Therefore this thesis proposes the usage of a central repository which should be externally supplied with the mapping rules by information model designers. Ideally these machine-readable mapping rules should also be written in a standardised way with a common language to express these rules. Although there isn't any existing standards for such mapping rules, Chapter 6 presents one possible way that was implemented within the scope of this thesis to demonstrate the concept.

5.1.2 Procedure for the Architecture development

The starting point for the development of the Software-Architecture, is the Architecture concept described in the previous section, in which the Meta-model was separated into a Producer-Consumer architecture 5.1.

Definition 5.1 ***Software Architecture***
Software Architecture of a system is defined as the structure or structures of the software components which make up the whole system. It encompasses the discipline of creating such structures and defines the relationships and properties of the software components of these structures [BCK03] [Cle+10]. Therefore a Software-Architecture defines the software components representing the system and how they interact with each other.

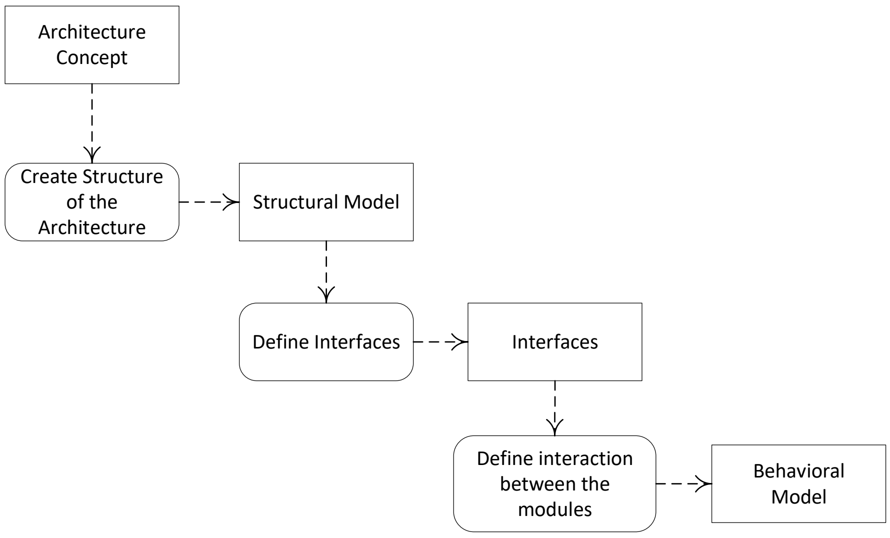


Figure 5.2: Procedure for the Architecture Development

In a Producer-Consumer architecture, the Consumers use the services offered by the Producer. Based on this, Section 5.2 defines a Software-Architecture in the form of a structural model. Based on the definition of the structural model and the Architecture concept, Section 5.3 defines the interfaces as well as the services, with the help of which the Consumers and the Producers will communicate with each other. The Software-Architecture is then refined according the dynamic requirements of the system thereby defining its behavior. Finally the behavioral model of the concept is developed in Section 5.4. The behavioral model along with the structural Model formulates the architecture for the Aggregated Integration Platform (Figure 5.2).

5.2 Structural Model

This sections deals with the structural model of the *Aggregated Integration Platform*. The structural model describes the components that make up the whole system as well as their relationships (as defined by the Software-Architecture). The subsection 5.2.1 depicts the components the platform comprises of. These have been depicted in the form of UML class diagrams and in some cases UML-Deployment diagrams to provide more clarity.

5.2.1 Conceptual Class Diagrams of the platform modules

The AIP and the AIC(s) together form the main components for the functionality of the Aggregated Integration Platform. The AIP and AIC(s) combined represent the meta-model that has been developed in Section 4.4, with the help of which Information Model-Management (4.1) activities are performed on the UIMs. The following subsections will organize the structural model of the platform by describing the aforementioned interfaces in the form of UML-classes and components.

Login-Module

The class *Login Services* implements the Login-Interface described in section 5.3.1. In order to authenticate a successful login the *Login-Services* class makes use of the *Access Control List*, which maintains a list of the authorized consumers and their respective access rights. The aforementioned list can be made in conjugation with the login credentials of the operating system hosting the application or can also be maintained in the form of a database that will be accessed at runtime for consumer authentication. Figure 5.3 depicts the class diagram of the Login Module.

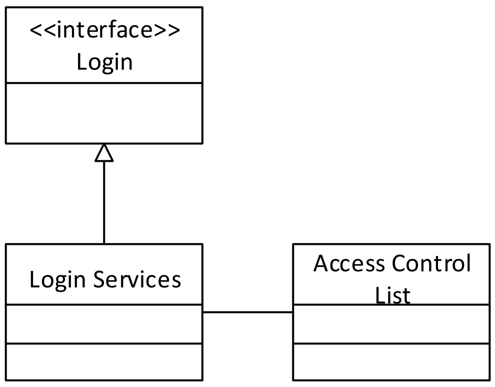


Figure 5.3: Login

AIM-Module

The access to all the objects modeled in the UIMs is provided through the *AIM* class. The AIM provides access to the *Instance Elements* and the *Type Elements* with the help of the *Basic Methods*. The class *AIM-Services* implements the AIM-Interface described in section 5.3.2. The AIM-Manager uses the services and provides access to the AIM. The AIM-Services class provides the needed access depending upon the access rights managed by the *Access* class.The AIM-Manager discovers new UIMs which can be aggregated, with the help of the *Discovery Manager* class which implements the discovery related services to find potential UIMs that can be aggregated. The *Configuration Manager* is responsible for providing the configuration information (for a UIM to be aggregated) to the AIM-Manager. The AIM-Manager uses the *Repo-Manager* class to retrieve the mapping rules and also the mapping information about the UIM components and their corresponding representations (in the AIM). The mapping rules are used during aggregation, and the one-to-one mapping of AIM and UIM components is used for reading, writing, subscribing etc. At this point it is important to consider the use of *Caching* for the AIP.

Definition 5.2 ***Caching***
Caching is the process of storing a copy of the data in a software or hardware component called Cache. This is done to provide faster retrieval of data for future requests of the same. The addition and removal of data from the Cache memory

depends on different Cache Management schemes [JNW10].

The principles of Caching can also be used in case of the AIP with a simple Cache management strategy. The first time a component in the UIM is accessed, it can be saved in the *AIM-Cache*, which is nothing but a deep-copy of the underlying component in the AIM [3]. When a consumer wants to access a component, it should also specify the *Age* which is acceptable for its use. This *Age* signifies the acceptable duration of the cached data. If the required *Age* is more than that of the Cached data, such a request can be served with a local copy of the underlying component, stored in the AIM. If the required *Age* however, is less than that of the cached data, the request has to be forwarded to the appropriate UIM and the response needs to be forwarded back to the requester. The Cache is managed by the Cache-Manager which is also responsible for refreshing the cached data as per a refresh-policy set by the AIP designer. When the data is to be acquired from the UIM, the *AIM-Manager* uses the *ACM-Manager*, which is then responsible to retrieve the data from the UIM and give it back to the AIP. Figure 5.4 represents the AIM and the classes that it interacts with.

[3] A deep copy in object oriented design is a copy of an object including all of its members i.e. new memory is assigned for the copied version of the object and the new memory locations are populated with the values from those memory locations which hold the object being copied [GR89].

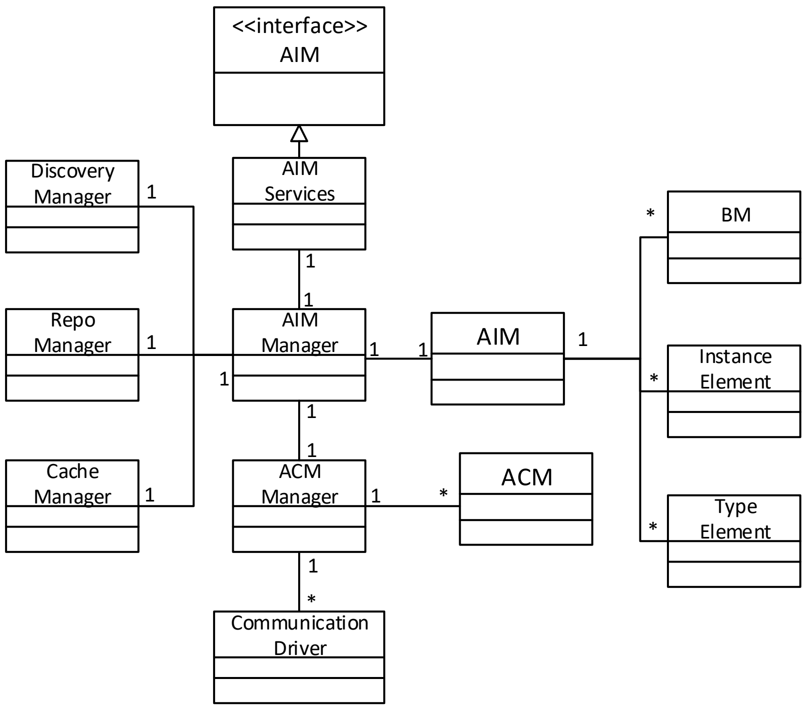


Figure 5.4: AIM

COM-Module

The class COM represents the consumer's view of the integrated information. It represents the GUI elements as well as the consumer specific business logic i.e the CBL elements. The relationship between the GUI elements to their respective functions is handled by the *COM Services* class, which implements the services mentioned in section 5.3.3. It has a relation to the Access class to verify the rights of a consumer to carry out a particular COM service. The *COM-Manager* class manages the different COM instances, as each defined use-case will be represented by an instance of the COM class.

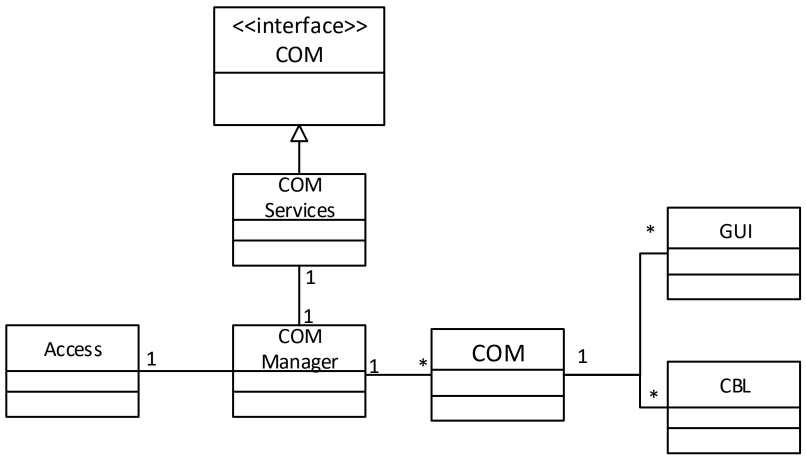


Figure 5.5: COM

ACM-Module

The class *ACM-Services* implements the ACM-Interface described in section 5.3.5. It provides the services to interact with the UIMs. In order to interact with the UIMs, different protocols might have to be followed. This requires the need of communication drivers, which are represented by instances of the ACM-Services class. The *ACM-Manager* class maintains a list of the required communication drivers. When a request to access an element from the UIM is received, the ACM-Manager recognizes the appropriate communication driver to be used and fetches the required information from the UIM using the identified communication driver. A class diagram of the ACM is shown in Figure 5.6.

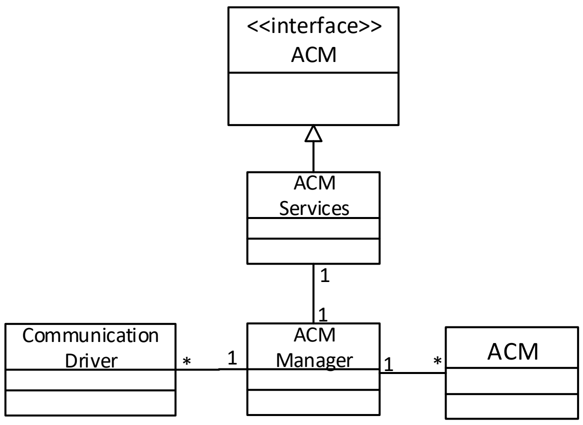


Figure 5.6: ACM

Configuration-Module

The class *Configuration-Services* implements the Configuration-Interface defined in section 5.3.6. It works in conjugation with the AIM while the process of aggregation is being carried out. The Configuration Services are used by the *Configuration-Manager* class, which manages the different *Configurations* of the different UIMs. The *Configuration* class also makes use of the *Security-Profiles* class to identify the security needs as specified by the aggregation requirements. A class diagram of the Configuration is shown in the figure 5.7.

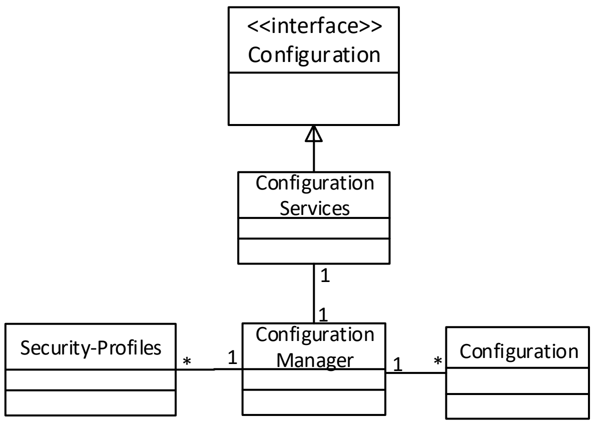


Figure 5.7: Configuration

Mapping-Module

The Mapping-Interface defined in 5.3.4 is implemented by the *Mapping-Repo* class. This class represents the mapping repository where the instance and type mapping rules will be saved. The class *Repo-Manager* manages the different mapping rules contained in the repository. While aggregating a UIM, The AIM uses the *Repo-Manager* to identify the mapping rules (if any) for the UIM being aggregated. The class *Rule Finder* is used by the Repo-Manager to identify the mapping rules associated with a particular UIM. The Repo-Manager is also responsible for the automatic creation of new rules in the repository. As described in the previous chapter 4.2.6, if we consider two information models A and B, IMR-AB represents the instance mapping rules and TMR-AB represents the type mapping rules. Now if the instance and type mapping rules between a third information model D and any one of the existing models (A or B) is introduced into the mapping repository, the *Rule Creator* should automatically generate the mapping rules between information model D and the remaining models (for which the rules have not been provided by the model designers). Figure 5.8 depicts an example scenario where IMR-AB and TMR-AB are already present in the repository and the rules between B and D (IMR-BD and TMR-BD) are then introduced into the repository. In this case the *Rule-Creator* will automatically create the rules between A and D, due to the transitive relationship between them. However this transitive relationship will only exist if the properties of an instance/type being compared are the same in the two mapping rules e.g. in the example shown in 4.8 in Chapter 4, no transitive relationship exists between the Mapping Rule A-B (properties being compared are Network Address in model A and Address in model B) and Mapping Rule A-C (properties being compared are ManufacturerID in model A and MakersID in model C).Therefore, IMR-AD and TMR-AD will be created and placed into the repository (5.7 (due to the existence of a transitive relationship between them) and an IMR-BC and TMR-BC will *not* be created due to the lack of transitivity. This means that if the model designers provide the mapping rules between a new information model and any of the models (the rules for which already exist in the repository), then the rules between the new information models and the rest of the models will automatically be generated by the *Rule-Creator* (if a transitive relationship exists between the rules). Figure 5.9 shows the class diagram of the Mapping module.

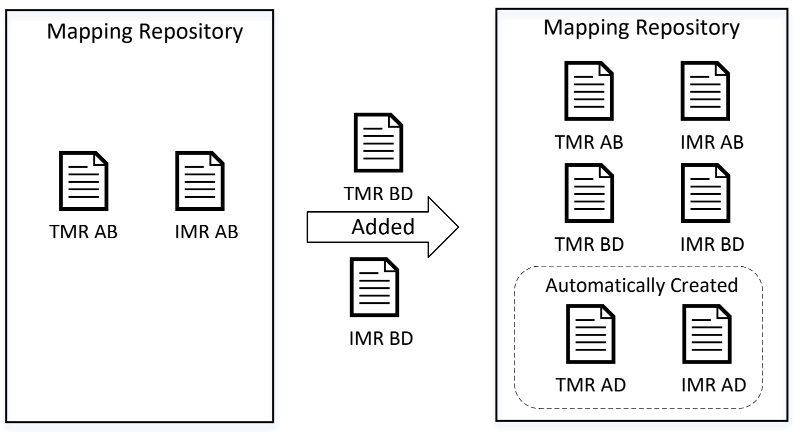


Figure 5.8: Automatic Rule Creation (based on a transitive relationship)

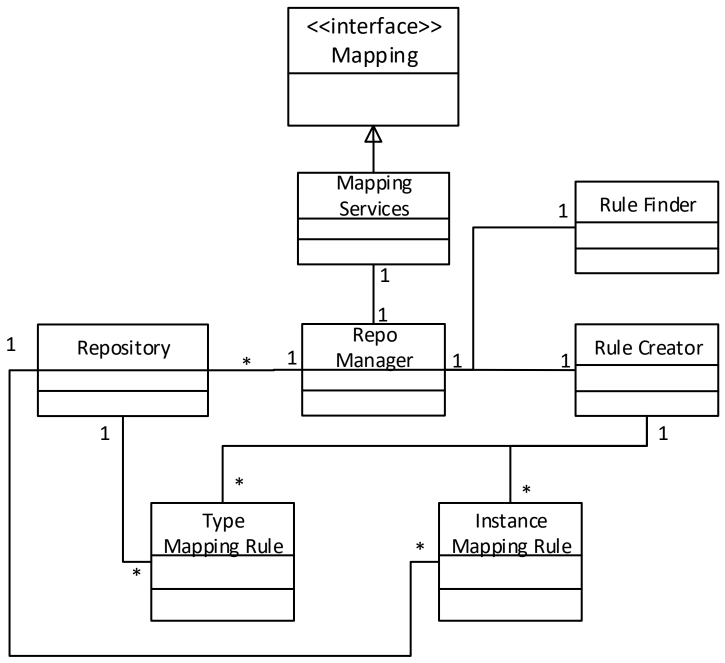


Figure 5.9: Mapping

Local and Global Mapping Repositories

The architecture defines a local and/or a global repository containing the instance and type mapping rules. Some information models can be proprietary and might not have a universal mapping rule with other standard information models. In such a case the mapping rules developed by the model designers, can be saved in a *local repository* where the AIP has been deployed. Also, when the mapping rules are being tested and not yet standardized, they can be saved in the *local repository.* Only when they are standard mapping rules between two standard information models, should they be saved in the *global repository.* This can be explained with the help of the deployment diagram shown in the Figure 5.10.

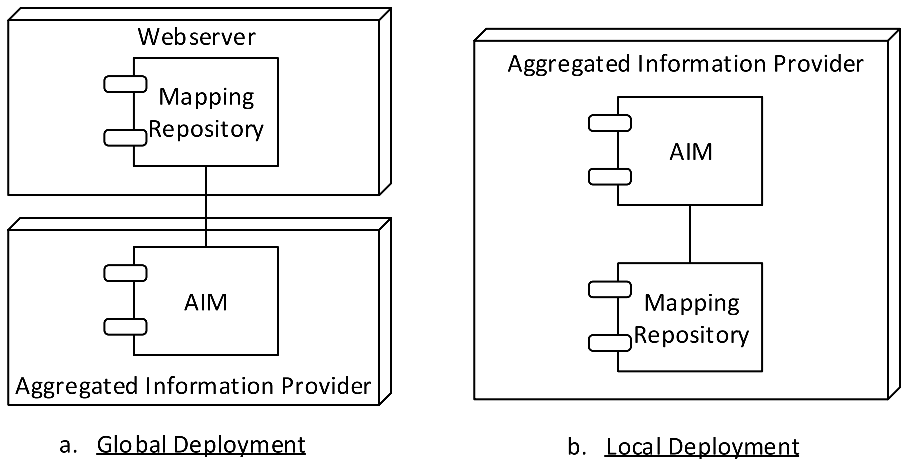


Figure 5.10: Deployment Diagram for Mapping Repositories

5.3 Definition of Interfaces

The main objective of the AIP is to provide the data and functions of the UIMs (aggregated by the AIM), to the Consumers in order to perform the activities identified by the use-cases defined in Section 4.2.1. In accordance to this, the interfaces required by the AIP can be defined in light of the outlined Requirements (Section4.2.2) and the observed use-cases.

Definition 5.3 ***Interface***
A Software Interface defines the services provided by a software component and encapsulates them within the component. It provides well defined entry points to allow access to the resources and services provided by a software component [PH04]

The following subsections describe the required AIP interfaces which will be used for communication with the Consumers on one side and with the UIMs on the other side. Figure 5.11 shows an overview of the interfaces.

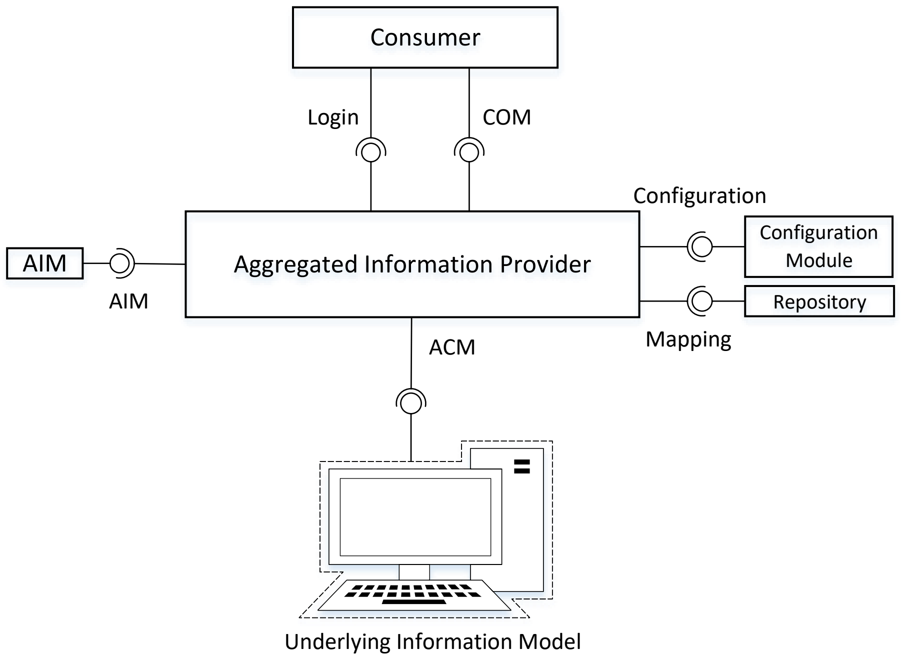


Figure 5.11: The Interfaces

The *Login-Interface* is used for authorization of the Consumer (Person and systems) in order to identify the User-Roles and access restrictions for that particular Consumer. The *COM-Interface* is the standard interface with the help of which the Consumer interacts with the AIM and can access the data and functions of one or more UIMs with the help of one or more use-case specific implementations of the *COM-Interface*. The *AIM-Interface* is the main interface which provides actual access to the data and functions of the aggregated UIMs. The *AIM-Interface* achieves this in co-operation with the *ACM-Interface*, the implementations of which take care of the communication-specific aspect of the information exchange between the AIM and the UIMs. The *Mapping-Interface* is specifically used to handle the access to the Instance and Type Mapping rules which involves the UIM concerned, and also to maintain a mapping between the UIM elements and their respective AIM elements. The management of the mapping rules is done with the help of the *Mapping-Interface*. The *Configuration-Interface* handles the configuration information about the UIMs, which are needed in order to establish a successful connection with the UIM and to be able to aggregate the model in the AIM. The following subsections will describe these interfaces and the services provided by these interfaces. These following descriptions assume a secure way of handling the communication between the different modules described in the previous section and does not explicitly refer to the security related aspects of these interfaces.

5.3.1 Login-Interface

The Login-Interface is used to oversee the role-based login and logout of the Consumers (Person or Systems). The function *Login* is used by the Consumer to gain access to the AIM specifying a specific Consumer-Role. The *Login* function needs the correct *LogidID* and *Password* for a successful login to the system to be able to access the AIM. The *Logout* function, as the name suggests, is used to logout of the system.

5.3.2 AIM-Interface

The AIM-Interface enables the consumer access to the data and functions provided by the UIMs. This access depends upon the access rights of the consumer. The function *GetUIMElement* provides read functionality to the appropriate Instance or Type element of the UIM. In order to be able to identify the relationships between the elements, the function *GetUIMRelationshipInfo* about an element can be used. This function should list out all the existing relationship that an element has with any other element in the information model. The write access (for a writable element) to the modeled elements would depend upon the access rights of the requesting consumer. When possible, the *WriteUIMElement* function can be used to write a value to the particular element. In order to keep the data in the UIM and the AIM consistent, a transaction mechanism must be implemented. This can be achieved with the help of the standard *BeginTransaction, EndTransaction and Commit* functions used for *Transaction Management* [Hen00]. In case of an incomplete transaction, the *Rollback* function restores the AIM or UIM to the previous consistent stage. The consumer can get notifications about value changes of a particular element by using the *SubscribeToUIMElement* function. The UIM will then generate events for the data changes and the consumer should receive it in a callback function *ElementChanged.* To call a *function* modeled as an *Instance-Element* in the UIM, the AIM offers a function called *InvokeUIMMethod* which can be used to pass input arguments to the *UIM-Method* and receive the returned result. The AIM provides the possibility to make changes to the UIM by adding or deleting an element with the help of the *AddUIMElement, DeleteUIMElement*

functions (if the consumer has the appropriate rights to do so).Similarly, the functions *UIMElementAdded and UIMElementRemoved* in AIM should be called by the UIM in case of elements being dynamically added or removed from it after it has already been aggregated in the AIM and the AIM should subsequently be refreshed using *RefreshAggregatedModel* function. Apart from these functions related to the UIM modeled elements, the *BrowseUIMElements* function provides a possibility to browse the whole structure of the UIM, element by element with the help of their relationships acting as links between the elements which the *browse* function can follow.
In terms of Aggregation of UIMs a few additional functions are provided by the AIM. The *DiscoverUIM* function is used to discover potential UIMs, which could be aggregated into the AIM. These discovered or pre-configured UIMs can be aggregated into the AIM with the *AggregateUIM* function.

5.3.3 COM-Interface

The COM-Interface is used by the consumer to access the AIM as per the specifications of the use-case. The COM-Interface also includes the graphical components to enable the consumer to interact with the aggregated model of the UIMs (as described in section 4.3.3). As discussed in Chapter 4, the COM provides additional analysis functions on the *aggregated* model, which are not intrinsically provided by the UIMs. The function *InvokeCOMFunction* can be used to call such a function, the algorithm for which, works on the collective data from multiple UIMs. This would also depend on the Access rights of the consumer. The COM-Interface also contains the companion functions which work in conjugation with the functions of the AIM-Interface to read and write data from and to the UIMs respectively. These functions include *GetAIMElement, SetAIMElement, SubscribeToAIMElement, AddAIMElement, DeleteAIMElement, BrowseAIMElements, GetAIMRelationshipElement etc.* Since to the COM, the UIMs will be practically invisible and it would only be interacting with the AIM. It is then the responsibility of the AIM to forward the request to the appropriate UIM and get the response from the UIM and forward it back to the consumer with the help of the COM.

5.3.4 Mapping-Interface

The Mapping-Interface takes care of all the mapping information required by the AIP. It helps to manage the *Instance and Type Mapping rules* in a *Mapping Repository.* It makes these mapping rules available to the AIM-Interface while aggregating an existing UIM in order to decide whether to add, merge or ignore the underlying instance or type element with respect to the AIM (Section 4.2.6). The functions *GetInstanceMappingRule* and *GetTypeMappingRule* are used to access the appropriate mapping rules (if available) from the mapping repository. The function *UploadMappingRule* is used to upload a new instance or type mapping rule into the local or global mapping repository. Furthermore, the Mapping-Interface is also used to maintain the mappings between the AIM and the UIM representations of the modeled elements. These mappings establish the links between the representation (of an UIM element) in the AIM to its underlying counterpart so that requests and responses can be forwarded accordingly between the consumer and the UIMs. The function *GetMappedUIMElementId* is used to get the ID of the UIM counterpart of an element in the AIM [4] and the function *GetMappedAIMElementId* works in the opposite direction and is used to get the ID of the AIM counterpart of an element in the UIM [5]. Apart from these the service *CreateNewMappingEntry* is used to create a new mapping entry between a UIM-element and an AIM-element.

5.3.5 ACM-Interface

The ACM-Interface provides the communication interface between the AIM and the UIMs. It deals with the conversion of the request and response formats expected by the UIM/AIM as defined by the communication protocol used by the UIM. The function *GetCommunicationDriver* is used to fetch the appropriate communication driver (for the said UIM) after the information about the appropriate UIM is provided by the Mapping-Interface. It contains all the protocol specific functions which are needed to interact with the modeled elements in the UIM *Connect,Disconnect, CreatSession, CloseSession, GetElement, SetElement, SubscribeToElement, AddElement, DeleteElement, BrowseElement, GetRelationshipInfo.* In case of a communication error, it provides the function *GetCommunicationError* in order to diagnose and rectify the cause of the communication error.

5.3.6 Configuration-Interface

The Configuration-Interface is used to manage the configuration related information about the UIMs. These include the protocol-specific addressing information, any credentials, and the necessary security related parameters (certificates,encryption algorithms etc.) needed to establish communication with the UIM. The function *LoadUIMConfiguration* is used to load the configuration artifacts of a pre-configured UIM. This helps to automatically aggregate the preconfigured UIMs, if they are available for aggregation. The *SaveUIMConfiguration* is used to save the UIM configuration to be made available automatically in case the aggregation process is re-started. The function *DeleteUIMConfiguration*

[4]Used to forward a request from the consumer to the appropriate UIM
[5]Used to forward a request from the UIM to the consumer

deletes the configuration related information about a UIM. Once deleted, configuration information needs to be provided again in order to aggregate the said UIM.

5.4 Behavioral Model

This section contains the dynamic model of the aggregation platform. The dynamic model refines the relationships between the classes defined in the static model by showing the interactions between them and the order of the same. Therefore this section discusses seven scenarios (derived from the functional requirements in 4.2.2) where the different modules interact with each other. When the consumer interacts with the AIM, the first step for the consumer is to always use the *Login* module in order to establish the access rights associated with it. This interaction has however, been only shown in section 5.4.1, but is to be assumed to be the first step also for sections 5.4.2, 5.4.3, 5.4.4 and 5.4.5.

5.4.1 Read Access to a UIM element

This section shows the read access to the UIM elements. Figure 5.12 shows the sequence diagram of the interactions between the different objects (of classes described in previous sections) when a consumer requests read access to a UIM element via the AIM. The process begins with the Consumer/COM [6] using the *Login* service to login successfully with the *username* and *password* which is then matched against the ACL to verify the validity of the same. This is followed by *Connect* (with valid security certificates), in order to access the AIM. Once the connection is successful, the AIM returns a connected *session* back to the consumer. This session will be used for the rest of the operations. The consumer can browse the AIM using the *BrowseAIMElements* service and when the consumer wants to read a particular element, it uses the *GetAIMElement* service of the AIM with the *ElementId* of the requested element.Upon receipt of this request, the AIM requests the Mapping module for the ID of the corresponding UIM element by using the service *GetMappedUIMElementId*. With this UIM element Id, the AIM-Manager sends the *GetUIMElement* service request to the *ACM Manager*. The ACM Manager then gets hold of the appropriate communication driver neeeded to communicate with the UIM with the *GetCommunicationDriver* service. Finally, with the help of the communication driver the ACM gets the value of the element using the *GetElement* service. The Data entity thus obtained is forwarded

[6]The consumer is based on the COM, so the the two terms consumer and COM have been used interchangeably in the following sections

in the reverse order from the UIM to the ACM to the AIM and finally back to the consumer.

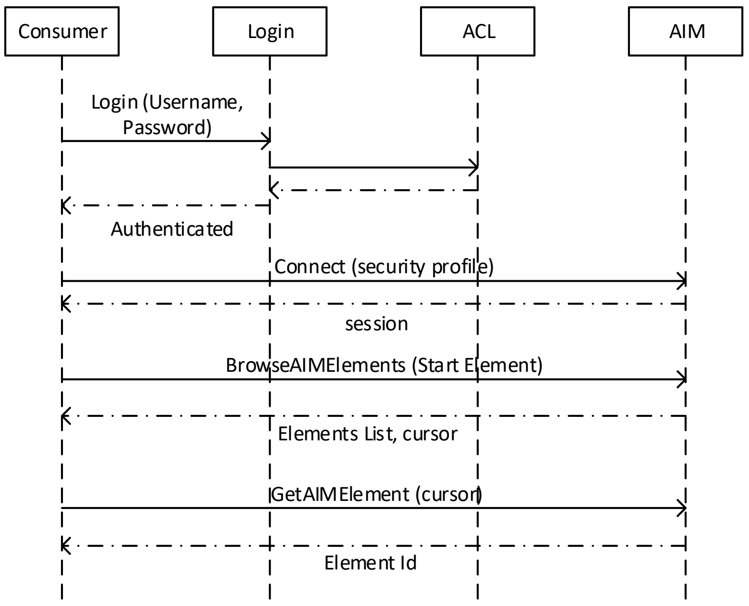


Figure 5.12: Getting the AIM element Id

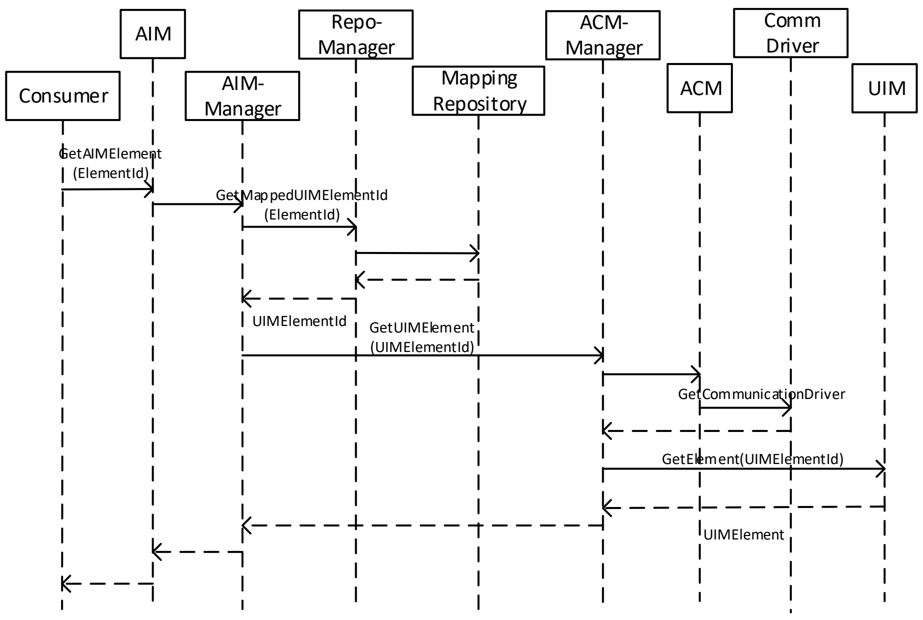


Figure 5.13: Reading a UIM element

5.4.2 Write Access to a UIM element

This section represents the write access to the UIM elements in the form of sequence diagrams similar to the read access explained in section 5.4.1. The same sequence (as the read access) is followed up until the ElementId of the required AIM element is obtained, if it is obtained while browsing the AIM. However, if the element id of the AIM element is already known, then directly after login and connect, the consumer can use the *SetAIMElement* service with the id of the element whose value is to be written, and the *Value* that needs to be written. Similar to the previous case, the AIM-Manager finds the corresponding UIM element Id with the help of the Mapping module using the service *GetMappedUIMElementId.* Upon receipt of the corresponding UIM element Id, the AIM-Manager uses the *SetUIMElement* service to send a write request to the ACM-Manager with the id and the *Value* (obtained from the SetAIMElement request). The ACM-Manager then writes the value to the UIM element with the help of the communication driver (obtained by using the *GetCommunicationDriver* service) and the *SetElement* service. The assumption in this depiction is that the consumer has the required access rights to change a value of the underlying element and the underlying element is indeed writable and is not read-only. The cases where a value cannot be written to an underlying element due to access limitations have not been shown in the following figures (5.14).

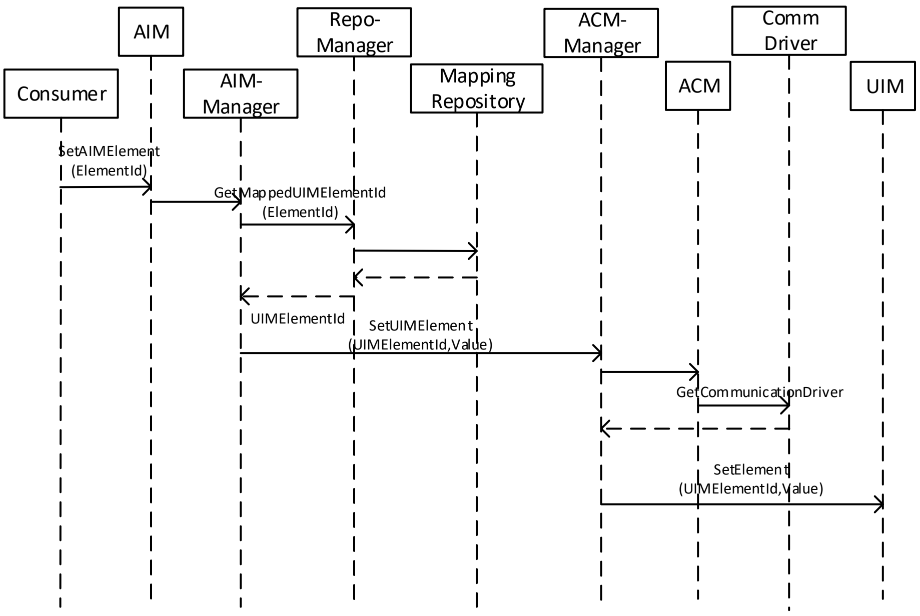

AIM
Repo-Manager
ACM-Manager
Comm Driver
Consumer
AIM-Manager
Mapping Repository
ACM
UIM
SetAIMElement (ElementId)
GetMappedUIMElementId (ElementId)
UIMElementId
SetUIMElement (UIMElementId,Value)
GetCommunicationDriver
SetElement (UIMElementId,Value)

Figure 5.14: Writing to a UIM element

5.4.3 Subscribing to a UIM instance

This section describes how subscriptions work in the aggregated environment with the help of sequence diagrams. In order to subscribe to an AIM element, the first thing the consumer needs is the AIM element id, which can be obtained either by browsing the AIM and selecting an element (as shown in section 5.4.1) or it has to known in advance. With the element ID, the consumer uses the *SubscribeToAIMElement* service. The AIM-Manager then finds the corresponding UIM element by using the *GetMappedUIMElementId*. With the UIM element Id, the AIM-Manager sends the *SubscribeToUIMElement* request to the ACM-Manager. The ACM-Manager caters to the request by obtaining the required communication driver (as in previous sections) and then by using the *SubscribeToElement* service via the communication driver. The UIM then returns the subscription id (Figure 5.15).

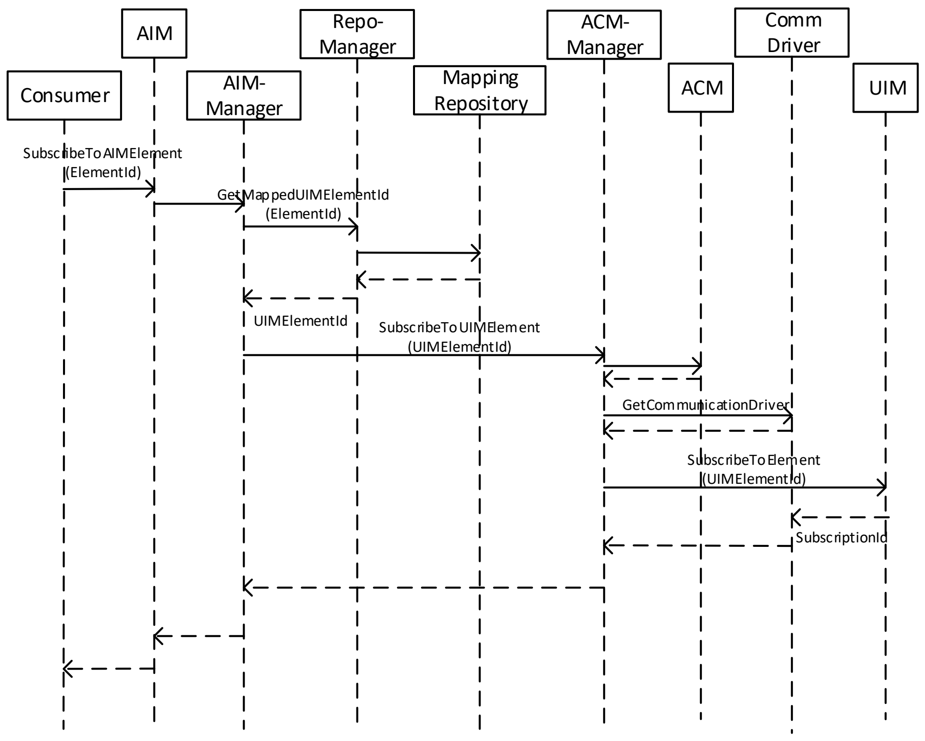

AIM
Repo-Manager
ACM-Manager
Comm Driver
Consumer
AIM-Manager
Mapping Repository
ACM
UIM
SubscribeToAIMElement (ElementId)
GetMappedUIMElementId (ElementId)
UIMElementId
SubscribeToUIMElement (UIMElementId)
GetCommunicationDriver
SubscribeToElement (UIMElementId)
SubscriptionId

Figure 5.15: Subscribing to a UIM element

After the subscription is successful, if the data value of the subscribed element in the UIM changes, it will send a corresponding *DataChangeEvent* to the ACM-Manager, containing the subscription id and the data changes. A data change means that the value of the UIM element (that was subscribed to) has been changed, therefore this change should trigger a value change of the corresponding AIM element, thus automatically triggering another *DataChangeEvent* with the subscription id for the subscription between the consumer and the AIM. Therefore when the *DataChangeEvent* is first received by the ACM-Manager, it finds the corresponding AIM element by using the *GetMappedAIMElementId* of the Mapping module. After getting the AIM element Id, the AIM-Manager writes the new (changed) value of the element to the AIM element by using the *SetAIMElement* service, to change the local (cached) value of the data entity. As soon as the value is changed, it should trigger a *DataChangeEvent* to the consumer[7]. Figure 5.16 shows the propagation of the *DataChangeEvent* from the UIM right upto the COM

[7]Alarms and Events can also be propagated in the same way from the UIM via the AIM and finally to the COM

(via the AIM).

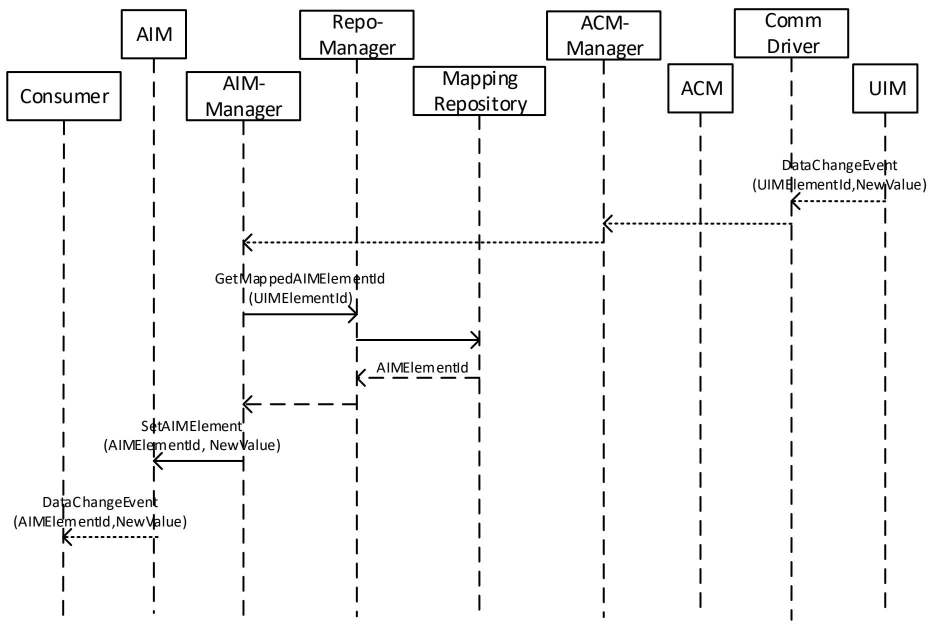

AIM
Repo-Manager
ACM-Manager
Comm Driver
Consumer
AIM-Manager
Mapping Repository
ACM
UIM
DataChangeEvent (UIMElementId,NewValue)
GetMappedAIMElementId (UIMElementId)
AIMElementId
SetAIMElement (AIMElementId, NewValue)
DataChangeEvent (AIMElementId,NewValue)

Figure 5.16: Subscription Data Change Event

5.4.4 Addition of a UIM element by Consumer

This section describes the sequence of interactions between the different modules when the consumer adds a new element to the AIM. In such a scenario, the newly added element, could be an element modeled directly in the AIM. If this is not the case, then it means that corresponding to this new element in the AIM, a new element should be added to one of the UIMs which the AIM aggregates. To which of the UIMs the new element would be added, should then be specified by the consumer (e.g. by providing a namespace). Apart from this, the consumer should also specify all the relationships that the newly added element in AIM would have with other modeled elements of the AIM. So the process is started by the consumer by using the service *AddAIMElement*, thus providing a *RelationshipList* of all the relationships it has with other AIM elements. This list will contain the AIM-element-ids of all the target elements (with which the relationship has to be formed). From each of the target elements, the AIM-Manager forms a list of the corresponding UIM-element-id by using the *GetMappedUIMElement* service of the Mapping module. After this, the AIM-Manager sends a *AddUIMElement* request to the ACM-Manager along with the newly compiled *RelationshipList* containing the target element ids in the UIM. With the knowledge about the UIM the new element is to be added to, the ACM-Manager finds the required communication driver by using the *GetCommunicationDriver* service. Then with the help of the appropriate communication driver, the ACM-Manager is able to add a new element into the UIM with the *AddElement* service, providing the *RelationshipList* with it . Once the new element is created in the UIM, the AIM-Manager requests the Repo-Manager to create a new map entry between the new AIM element and the new UIM element using the CreateMapEntry service(Figure 5.17).

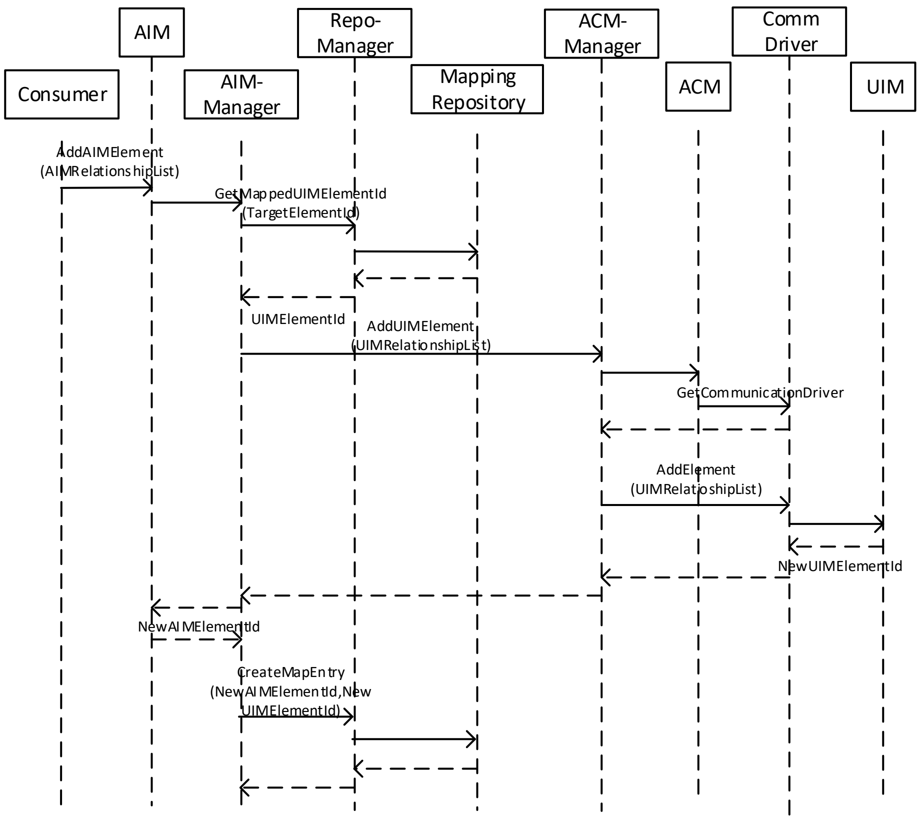

AIM
Repo-Manager
ACM-Manager
Comm Driver
Consumer
AIM-Manager
Mapping Repository
ACM
UIM
AddAIMElement (AIMRelationshipList)
GetMappedUIMElementId (TargetElementId)
UIMElementId
AddUIMElement (UIMRelationshipList)
GetCommunicationDriver
AddElement (UIMRelationshipList)
NewUIMElementId
NewAIMElementId
CreateMapEntry (NewAIMElementId,New UIMElementId)

Figure 5.17: Addition of an element by the consumer

5.4.5 Deletion of a UIM element by Consumer

This section describes the sequence of interactions between the different modules when the consumer deletes an element from the AIM. In this case the consumer first sends a *DeleteAIMElement* request to the AIM-Manager, with the AIM-element-id of the element to be deleted. The AIM-Manager the finds the corresponding UIM-element-id by using the *GetMappedUIMElement* service of the Mapping module. After getting the UIM-element-id, the AIM-Manager sends the *DeleteUIMElement* request to the ACM-Manager, providing the UIM-element-id with its request. The ACM-Manager fulfills the request by first getting the required communication driver by using the *GetCommunicationDriver* service and then using the driver to call the *DeleteElement* service . Once the element is deleted in the UIM, the AIM-Manager requests the Repo-Manager to delete the map entry corresponding to the deleted AIM element and the deleted UIM element using the DeleteMapEntry service(5.18)).

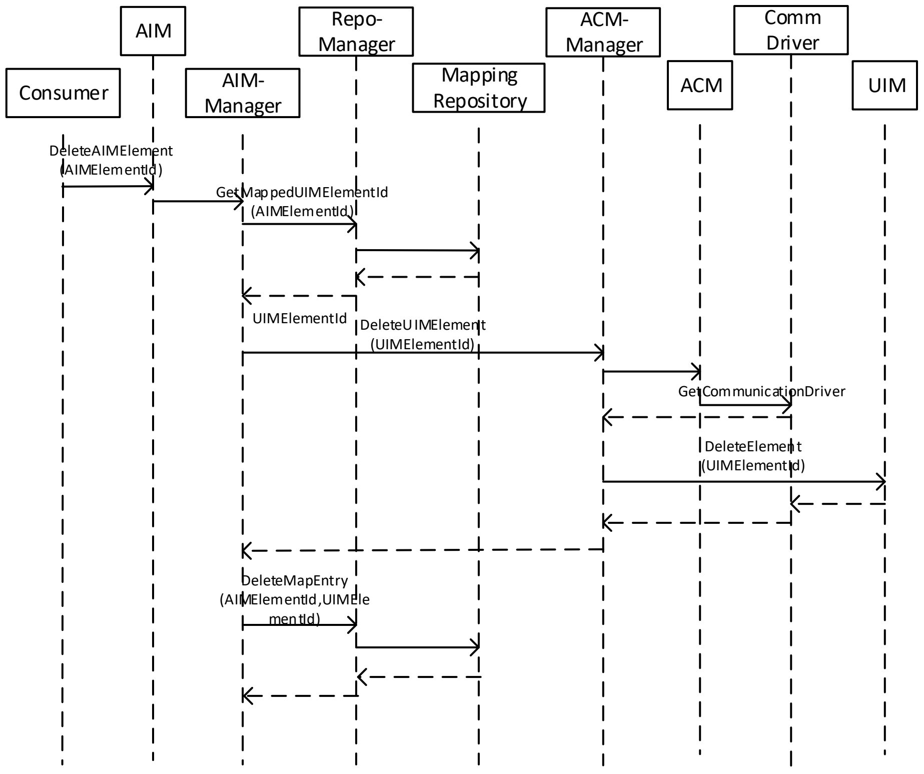

AIM
Repo-Manager
ACM-Manager
Comm Driver
Consumer
AIM-Manager
Mapping Repository
ACM
UIM
DeleteAIMElement (AIMElementId)
GetMappedUIMElementId (AIMElementId)
UIMElementId
DeleteUIMElement (UIMElementId)
GetCommunicationDriver
DeleteElement (UIMElementId)
DeleteMapEntry (AIMElementId,UIMElementId)

Figure 5.18: Deletion of an element by the Consumer

5.4.6 Dynamic Addition of a UIM element

This section describes the sequence of interactions between the different modules when a new element is added directly to a UIM (e.g. a plug and play device plugged in, thereby creating a new element in an existing information model). This again, is an example of propagation of an event from the UIM up to the consumer, which in this case is a *InformationModelChangeEvent* generated by the UIM. This is based on the principle that when a new element is added to an already existing (running) information model, a corresponding new element should also be added to the AIM (which aggregates the UIM). Therefore when a new element is added to the UIM, an *InformationModelChangeEvent*, containing the details of the newly added element and its relationships with other UIM-elements, is generated and sent to the ACM-Manager via the appropriate communication driver. The ACM-Manager forwards the event to the AIM-Manager. The AIM-Manager upon getting the event, creates a new AIM-element along with its relationships [8]corresponding to this new UIM-Element and creates the new mapping information in the Mapping Module by using the service *CreateNewMapEntry* service (Figure 5.19).

[8]the AIM-elements corresponding to each of the relationship target elements in the UIM are found out with the help of the Mapping module and then each relationship is then correspondingly established

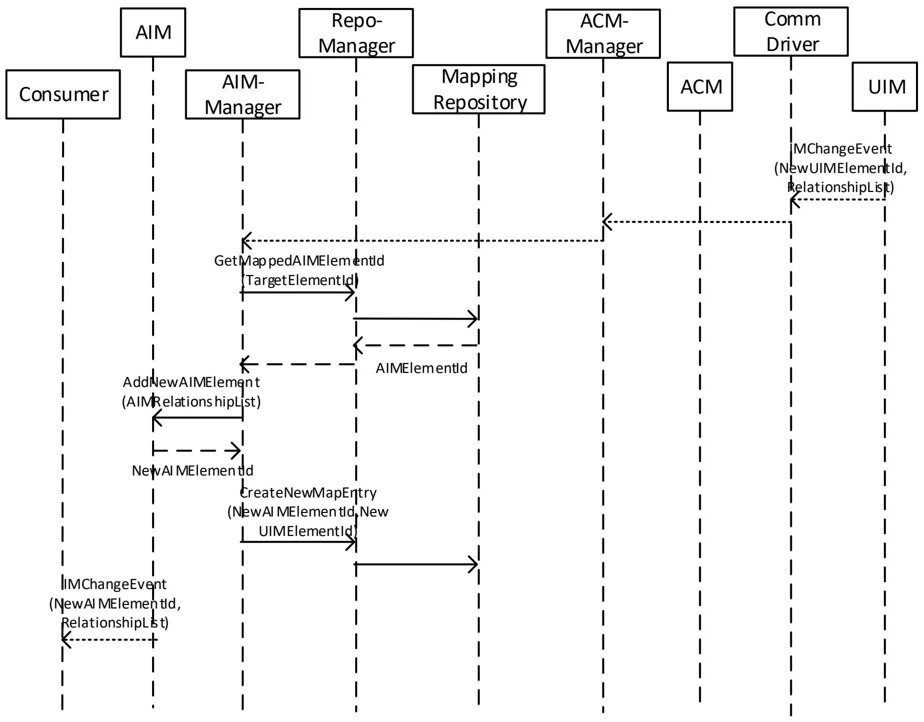

Consumer
AIM
AIM-Manager
Repo-Manager
Mapping Repository
ACM-Manager
ACM
Comm Driver
UIM
IMChangeEvent (NewUIMElementId, RelationshipList)
GetMappedAIMElementId (TargetElementId)
AIMElementId
AddNewAIMElement (AIMRelationshipList)
NewAIMElementId
CreateNewMapEntry (NewAIMElementId, NewUIMElementId)
IMChangeEvent (NewAIMElementId, RelationshipList)

Figure 5.19: Dynamic Addition of a UIM element

5.4.7 Dynamic Deletion of a UIM element

This section describes the sequence of interactions between the different modules when an existing UIM-Element is deleted from the UIM directly (not by the consumer). Similar to section 5.4.6, the dynamic deletion of a UIM element, will also change the UIM structure. Therefore even in this case, a *InformationModelChangeEvent* will be generated by the UIM, containing the details about the deleted UIM-element. The ACM-Manager listens for this event using the communication driver dedicated for the involved UIM. Upon receipt of this *InformationModelChangeEvent*, the ACM-Manager forwards it to the AIM-Manager. The AIM-Manager finds the AIM-element corresponding to the deleted UIM-element with the help of the *GetMappedAIMElement* service of the Mapping module. The AIM-Manager then deletes the corresponding AIM-element and its relationships from the AIM using its *DeleteAIMElement* service. Once the element is deleted in the AIM, the AIM-Manager requests the Repo-Manager to delete the map entry corresponding to the deleted AIM element and the deleted UIM element using the DeleteMapEntry service (5.20).

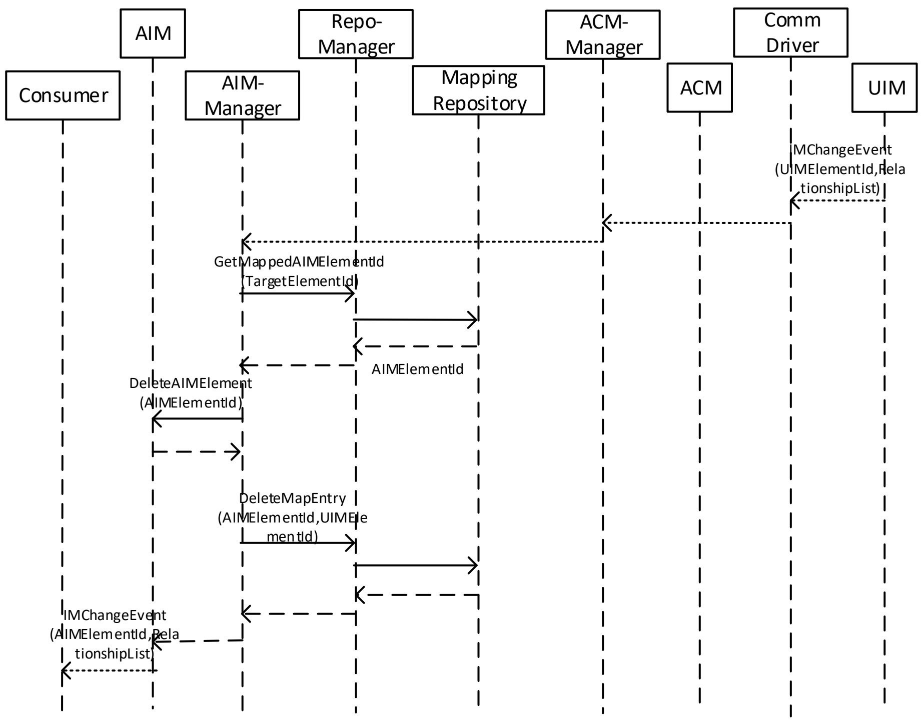

AIM
Repo-Manager
ACM-Manager
Comm Driver
Consumer
AIM-Manager
Mapping Repository
ACM
UIM
MChangeEvent (UIMElementId,RelationshipList)
GetMappedAIMElementId (TargetElementId)
AIMElementId
DeleteAIMElement (AIMElementId)
DeleteMapEntry (AIMElementId,UIMElementId)
IMChangeEvent (AIMElementId,RelationshipList)

Figure 5.20: Dynamic Deletion of a UIM element

5.5 Summary

This chapter develops the Architecture for the Aggregated Integration Platform. Section 5.1 introduces the basic Architecture concept in the form of a Producer-Consumer architecture. Section 5.2 describes the structural model of the Aggregated Integration Platform where it shows the class diagrams of the different components in the architecture. The following section 5.3 describes the required interfaces for the development of the architecture. Section 5.4 describes the Behavioral model of the platform, so as to show the dynamic interactions between the different modules of the platform in the form of sequence diagrams involved in seven example scenarios which correspond to use-cases for the aggregation platform. These concepts discussed in this chapter will form the basis for Chapter 6 of this thesis.

Chapter 6

Realization and Review

Contents

6.1 Overview of the Implementation 139

6.1.1 Overview of the Prototype 140

6.1.2 Process of Prototype Implementation 141

6.2 Implementation Technologies 142

6.2.1 Description of Mapping Rules 143

6.2.2 Client-Server Technology 143

6.3 Illustration of the aggregated platform 143

6.3.1 Underlying Servers 144

6.3.2 Aggregation Server 147

6.3.3 Configuration 150

6.3.4 Consumer Client 150

6.3.5 Mapping Rules Repository 151

6.4 Evaluation of the implemented prototype 155

6.5 Summary 156

6.1 Overview of the Implementation

This section provides the overview about the implementation of the prototype which was developed in order to demonstrate the Aggregated Integration Platform concept outlined in this thesis. Within this section, at first an overview about the prototype itself has been provided in Section 6.1.1 which includes the typical scenarios considered for the prototype implementation. The following section 6.1.2

describes the procedure used during the implementation of the introduced concept paying special attention to the implementation technologies used for the management of mapping rules as well as the client-server architecture used for the development of the integration platform.

6.1.1 Overview of the Prototype

The architecture of the Aggregated Integration Platform was explained in Chapter 5. The structural and behavioral model introduced in the previous chapter was implemented with the help of an OPC UA based client-server architecture. The aggregated integration platform itself can be viewed as a combination of an Aggregation Server (with an embedded OPC UA server exposing the AIM and several embedded OPC UA clients) and an OPC UA Client (representing the COM) connected to the Aggregation Server. A simplified overview of the prototype is shown in figure 6.1, whereby the Aggregation Server consists of several dedicated clients C1, C2 and C3 connected to the underlying servers US-PADIM, US-DEXPI and US-40223. This prototype aims to implement the typical aggregation scenario mentioned in section 4.2.4. Therefore the underlying servers US-PADIM, US-DEXPI and US-40223 have been implemented to expose practical information models based on the PADIM, DEXPI and OPC40223 models respectively. The Aggregation Server is the AIP mentioned in the previous chapter (5.2.1) which hosts the AIM and communicates with the Mapping repository (handling the mapping rules). For this particular prototype a generic AIC has been chosen, which is capable of providing the basic consumer functionality on the AIM e.g. Reading, Writing and Subscribing of data elements exposed in the AIM. C1, C2 and C3 are implementing the ACMs mentioned in the previous chapters and are responsible of communicating with the respective underlying servers. Further implementation details about the prototype have been covered in the forthcoming sections.

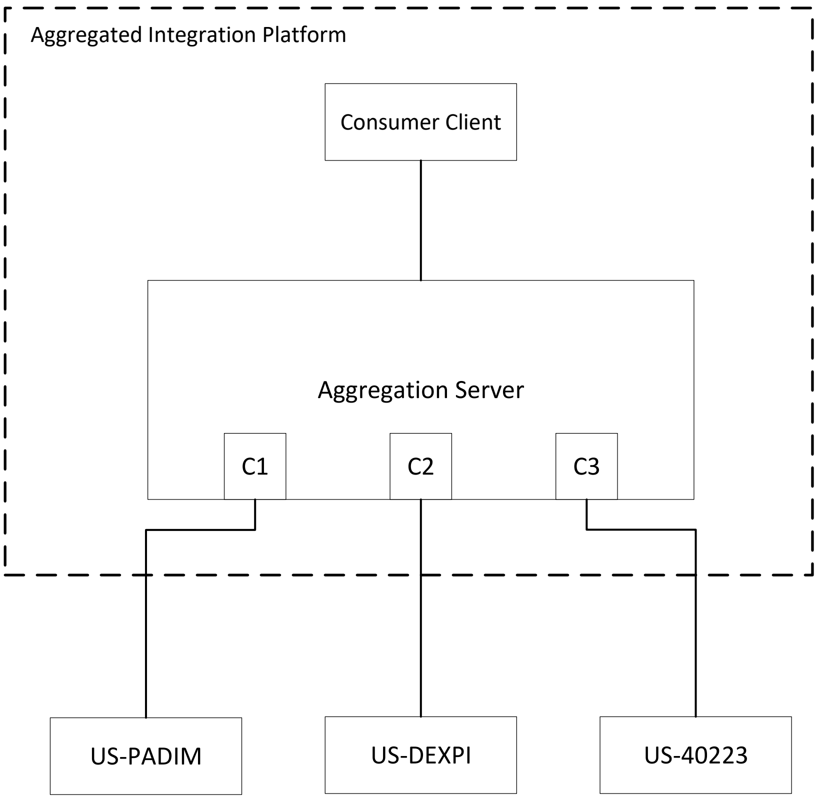


Figure 6.1: Overview of the prototype

6.1.2 Process of Prototype Implementation

The prototype development process closely follows the V-Model introduced in 3.1.1. The software architecture mentioned in 5.1 and the handling of mapping rules for the merging of information models (4.2.6) are the starting points for the realisation of the prototype. Therefore there are mainly two independent parts to this prototype namely the management of the mapping rules which are needed to solve the overlaps in the information models and the client-server architecture to manage the interaction with the resulting AIM. In conclusion, after finalising the implementation technology to describe the mapping rules, two instance mapping rules were written i.e. one that maps a instance representing the same pump (from the example in 4.2.4) between the DEXPI and PADIM information models (as per 4.2.7), and another mapping rule that provides this mapping between the PADIM and OPC40223 models. Similarly the implementation technology for the development of the client-server architecture was chosen as per the requirements outlined in 4.2.2 and the AIM was developed considering the aforementioned mapping rules and was hosted by an aggregating server. In the end the implementation was validated against the requirements. The different modules of the prototype will be detailed in the forthcoming sections along with the reasoning behind choosing the implementation technologies.

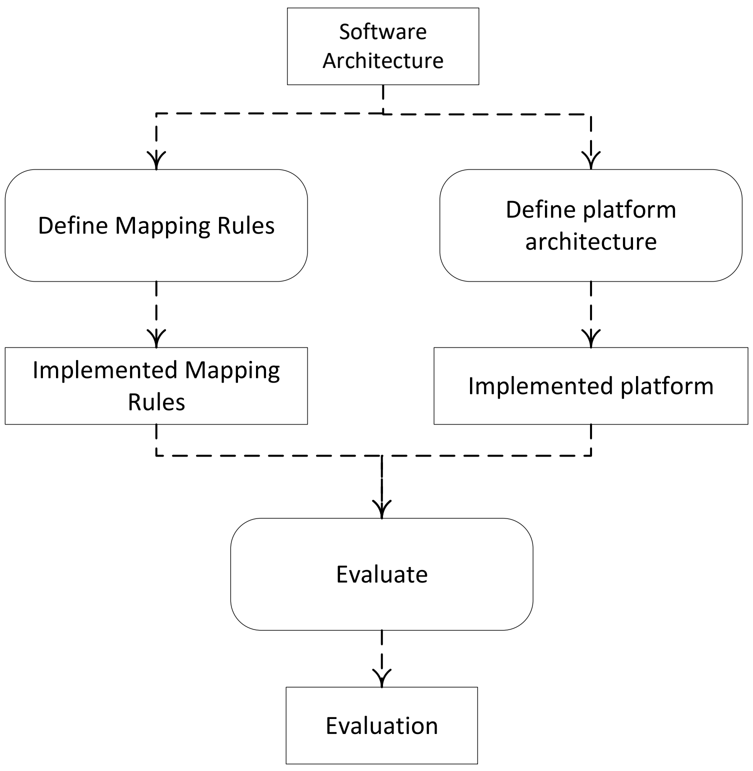


Figure 6.2: Process of Prototype Implementation

6.2 Implementation Technologies

As mentioned in the previous section, up until now the software architecture and the mapping rule descriptions form the basis for the development of the aggregated platform. A suitable implementation technology needs to be chosen to describe the mapping rules. Similarly the client-server architecture also needs to be implemented choosing the technology that suits best with respect to the functional and non-functional requirements detailed in 4.2.2. This section describes the points considered to reach an informed decision about the implementation technology to be used.

6.2.1 Description of Mapping Rules

For the description of the mapping rules, two ways to represent the information were considered namely JSON and XML. For the purposes of this thesis, JSON was chosen to describe the mapping rules between the pairs of information models i.e. a JSON file mapping the pump in our example scenario between the DEXPI and PADIM models and another JSON file providing the rule to map the instances representing the pump between the PADIM an OPC40223 models. Table A.1 in the Annex section shows a comparison between JSON and XML supporting the usage of JSON for this particular application as this is a relatively simple application where ease and speed of use are more important.

6.2.2 Client-Server Technology

The Producer-Consumer design pattern mentioned in the previous chapter (5.1.1) was implemented with a client-server architecture whereby the AIP consists of a server hosting the AIM and three clients connected to each of the underlying servers, each hosting the three UIMs. As per the discussion in 3.3 OPC UA was chosen as the implementation technology for the client-server architecture. Not only does OPC UA fulfil most of the functional and non-functional requirements mentioned in 4.2.2, there are already OPC UA Companion Specifications (3.4.3) available for the PADIM (OPC 30081 - Process Automation Devices) [Fou21a] and DEXPI (OPC 30250 - UA Companion Specification for DEXPI) [Fou21c] information models (the OPC 40223 is an intrinsic OPC UA Companion Specification for Pumps and Vacuum Pumps [OF21]).

6.3 Illustration of the aggregated platform

This section will detail the different modules implemented to illustrate the working of the aggregated platform as a proof of concept demonstration of the concepts introduced in this thesis. The different modules have been illustrated in 6.3. The following sections will also relate each of the implemented modules to the structure model of the platform from 5.2. The implementation of the underlying servers and the aggregating server was done using the Unified Automation .NET SDK.

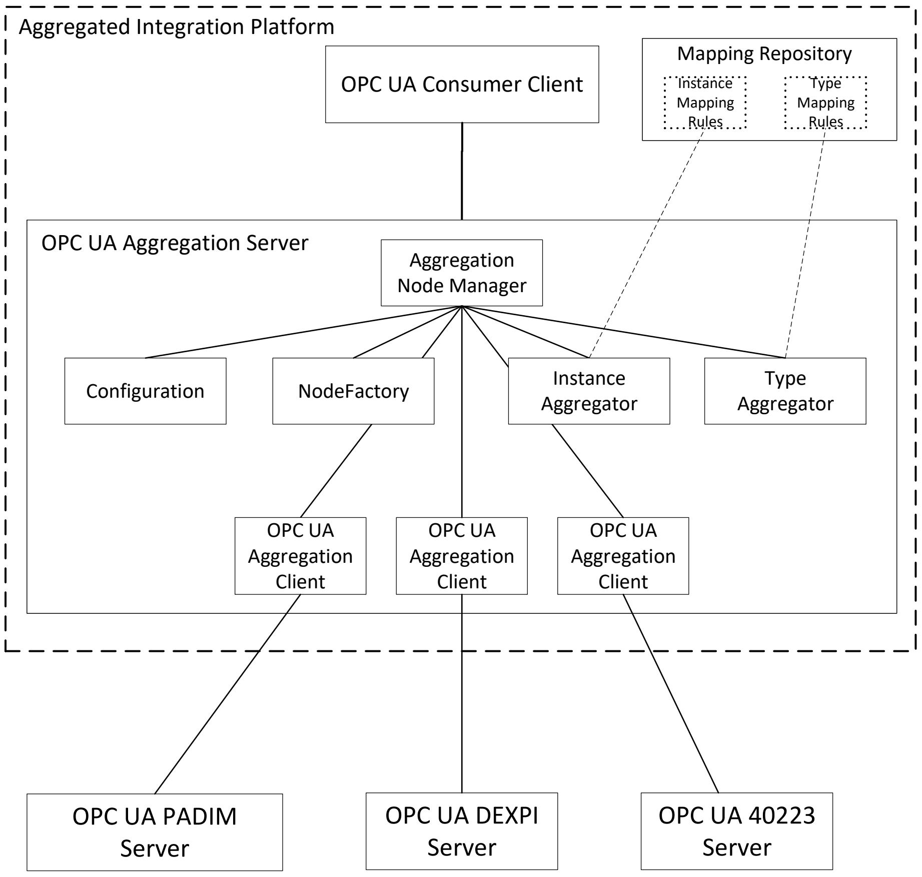


Figure 6.3: Implemented modules in the aggregated platform

6.3.1 Underlying Servers

Three OPC UA servers (acting as underlying servers in the example scenario) were developed for this demonstration, each hosting a UIM for the PADIM, DEXPI and OPC 40223 models respectively. A fictitious pump was modelled in all three servers respectively using the OPC UA *TypeDefinitions* from the three companion specifications i.e. in the PADIM OPC UA Server, an instance of the *PADIMType* was created, in the DEXPI OPC UA Server an instance of the *CentrifugalPumpType* was created and finally in the OPC 40223 Server an instance of the *PumpType* was created. The respective properties of these instances were populated as per the example shown in 4.2.7. These instances have been represented according to OPC UA notations in the following figures 6.4,6.5 and 6.6 (not all properties have been shown in the figures for the sake of simplicity). The address space in their respective OPC UA servers can be found in Annex (A.1, A.2, A.3).

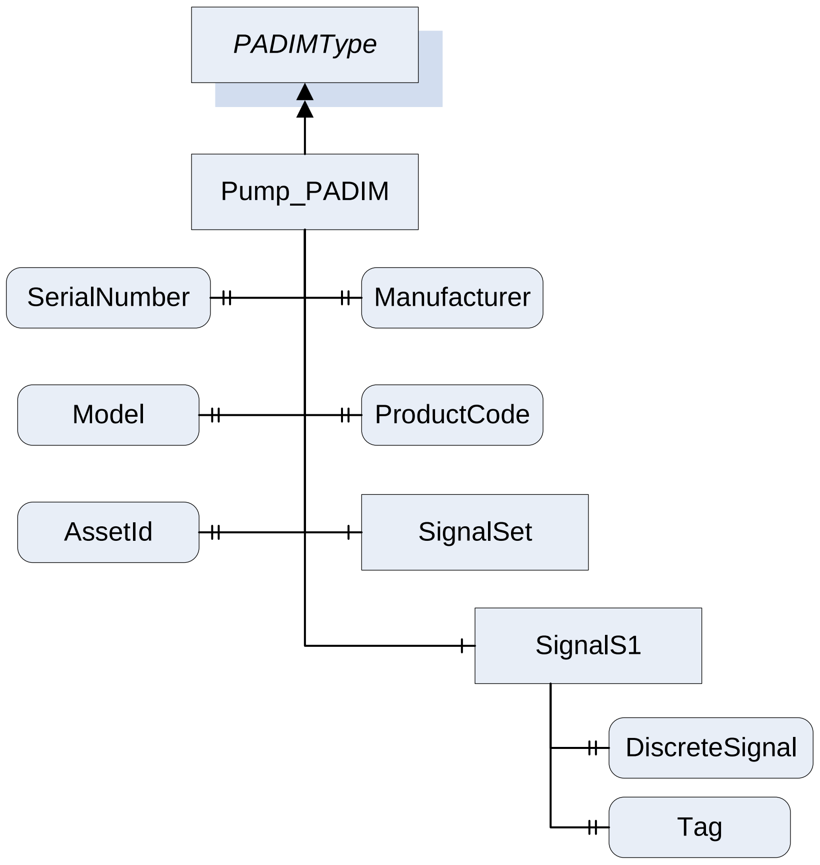


Figure 6.4: Pump instance in the PADIM OPC UA Server

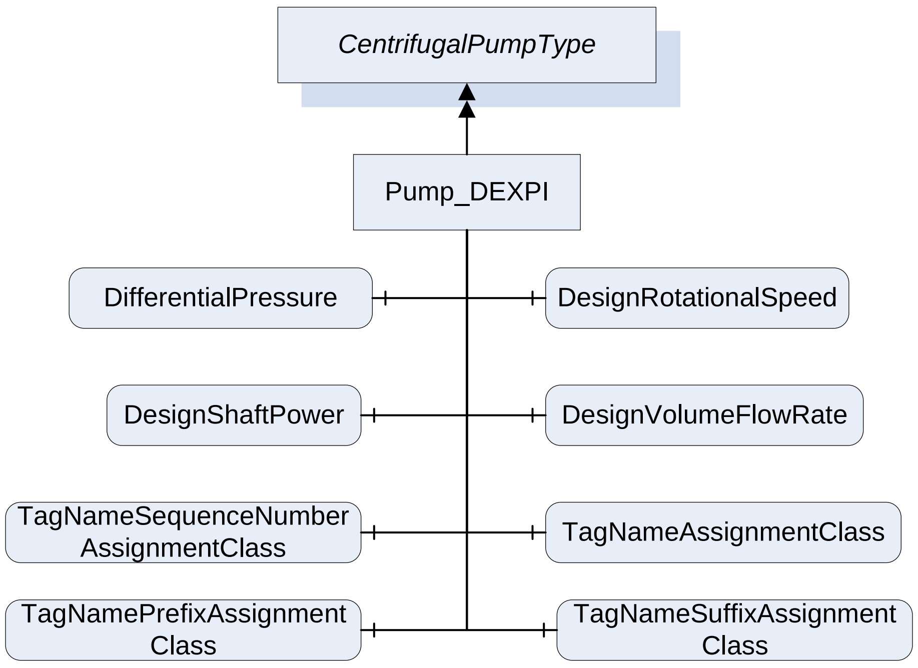


Figure 6.5: Pump instance in the DEXPI OPC UA Server

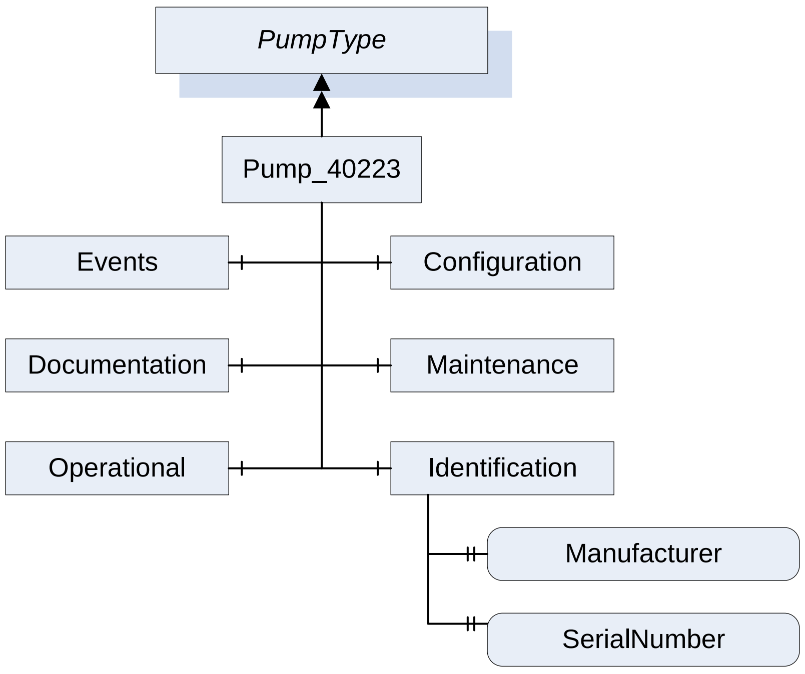


Figure 6.6: Pump instance in the OPC 40223 OPC UA Server

6.3.2 Aggregation Server

The Aggregation Server is the core of this thesis and implements the main aggregation concepts introduced in the previous chapters.The Aggregation Server represents the AIP introduced in section 5.1.1 and hosts the AIM in its address space. As can be seen from figure 6.3, the Aggregation Server contains several interconnected components connected to each other. Each of these components will be discussed in the following subsections.

Aggregation Node Manager

The Aggregation Node Manager implements the AIM-Module of the platform (5.2.1). It manages the address space of the Aggregation Server i.e. it manages all the nodes present in the AIM and communication with the same. It acts as a middleware between the Consumer Client (6.3.4) and the three underlying servers (6.3.1). It creates the nodes in the aggregated address space by browsing the address spaces of the three underlying servers with the help of three corresponding Aggregation Clients (6.3.2) connected to each of the three underlying servers respectively. For each node in the underlying servers, a corresponding node is present in the aggregated address space. Since the underlying servers are the actual sources of information for the Aggregation Server, it maintains mapping dictionaries between the nodes in its own address space and the corresponding nodes in the underlying servers. When the Consumer Client requests the read-/write/subscribe services for a particular node in the aggregated address space, the Aggregation Server forwards these requests to the corresponding node from one of the underlying servers (using the corresponding Aggregation Client). Similarly it forwards the responses from the underlying server to the Consumer Client.

Type Aggregator

The Type Aggregator is used by the Aggregation Node Manager to create the *Type* nodes of the Aggregation Server. During the aggregation process, when the Aggregation Node Manager encounters a *Type* node in the underlying server, it uses the Type Aggregator which in turn uses the *Type Mapping Rules* to decide if the *Type* node from the underlying system already exists in the Aggregation Server address space. If the *Type* node doesn't exist in the address space of the Aggregation Server, a new *Type* node is added to it. If as per the *Type Mapping Rules*, the same or similar type already exists in the Aggregation Server address space, no new *Type* node is added, however the mapping dictionaries are updated to point to the same or similar *Type* node.

Instance Aggregator

The Instance Aggregator is used by the Aggregation Node Manager to create the *Instance* nodes of the Aggregation Server. During the aggregation process, when the Aggregation Node Manager encounters a *Instance* node in the underlying server, it uses the Instance Aggregator which in turn uses the *Instance Mapping Rules* to decide if the entity represented by the *Instance* node from the underlying system already exists in the Aggregation Server address space. If an *Instance* node representing the entity doesn't exist in the address space of the Aggregation Server, a new *Instance* node is added to it. If as per the *Instance Mapping Rules*, an *Instance* node representing the same entity already exists in the Aggregation Server address space, a new *Instance* node of *Type AggregatedInstanceType* is created in the Aggregation Server address space. Thereafter, another *Instance* node representing the *Instance* node of the underlying server is added to the Aggregation Server address space and both the *Instance* nodes (new and previously added) in the Aggregation Server address space are reorganized as *Child* nodes of the aforementioned *Instance* node of *AggregatedInstanceType*. The mapping dictionaries are updated accordingly. At the end of the aggregation process, all *Instances* from the underlying serveres representing the same entity are aggregated under a single *Instance* of *AggregatedInstanceType* within the Aggregation Server address space. In the case of the prototype implementation, the instances representing the same pump i.e. *Pump_PADIM* in the underlying PADIM server, the *Pump_DEXPI* in the underlying DEXPI server and the *Pump_40223* in the underlying OPC 40223 server were all aggregated under a single *AggregatedPump* instance in the Aggregation Server. The *AggregatedInstanceType* has a property called *AggregatedModels* which contains a list of all the models that have been aggregated under the said aggregated instance in the Aggregation Server. The OPC UA representation of this *AggregatedPump* instance is shown in Figure 6.7.

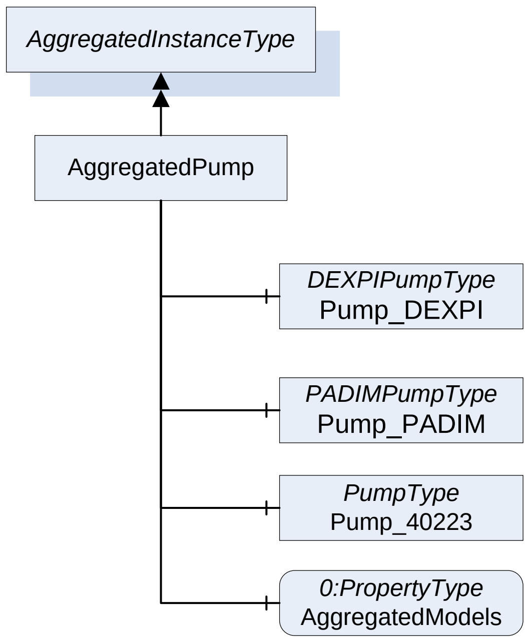


Figure 6.7: Aggregated Pump instance in the Aggregation OPC UA Server

Node Factory

In an OPC UA Address space, each node has a unique server-wide *NodeId* which serves as an identifier for the node. When the underlying servers are being aggregated by the Aggregation Server, when new nodes need to be added to the address space of the Aggregation Server, the Node Factory creates these nodes (with their unique *NodeId*). Moreover all nodes in an OPC UA Address space belong to a so-called *NodeClass*. A node might represent an *ObjectType*, a *ReferenceType*, an *Object* (instances of *ObjectTypes*), a *Variable*, a *Method* or a *DataType*. The *NodeClass* defines what *Attributes*, *References*, *Properties* can be expected from a particular node (the *NodeId* is one such *Attribute*). Therefore during the aggregation process the *NodeFactory* creates a node of the appropriate *NodeClass* depending upon the *NodeClass* of the node in the underlying server that is being aggregated.

Aggregation Clients

As mentioned in 6.3.2, the Aggregation Node Manager communicates with the underlying servers with the help of Aggregation Clients connected to each of the underlying servers. The Aggregation Clients implement the ACM-Module of the platform(5.2.1). In the prototype implementation, three OPC UA Clients were created for the three underlying servers. The mapping dictionaries of the Aggregation Node Manager also contain information about the client sessions a particular node in the address space of the Aggregation Server is associated with. This means that when the Consumer Client tries to access a particular node, the Aggregation Node Manager consults the mapping dictionaries and finds out the appropriate Aggregation Client that can then access the appropriate node in the underlying server and serve the request.

6.3.3 Configuration

The Configuration component implements the Configuration-Module 5.2.1 of the platform. It is used to store the parameters needed to create a connection session with the underlying servers. This includes the endpoint descriptions, the security profiles and the login information needed in order to connect to the underlying servers. This information is already available before the Aggregation Server is started up. During the aggregation process, the Aggregation Node Manager uses this information to create the Aggregation Client sessions with the respective underlying servers. In the prototype implementation, this component has been implemented as an XML file storing the aforementioned information about the underlying servers, which is loaded by the Aggregation Node Manager upon startup of the Aggregation Server.

6.3.4 Consumer Client

The Consumer OPC UA Client implements the Login-Module (5.2.1) and the COM-Module (5.2.1) of the platform. It connects to the Aggregation Server and uses the OPC UA Services (*Read, Write, Subscribe etc.*) offered by the Aggregation

Server. Since the prototype does not deal with any particular use-case for the consumption of the aggregated information, the was no specific business logic to be implemented in the Consumer Client. Therefore in the prototype implementation the Consumer Client is represented by the UAExpert Generic OPC UA Client as it provides a possibility to test most of the functional requirements (4.2.2) for the platform.

6.3.5 Mapping Rules Repository

This component implements the Mapping-Module of the platform (5.2.1). To realise the repository concept introduced in 5.2.1, two instance mapping rules were written using JSON. Since the underlying servers are OPC UA servers, the mapping rules have been kept relatively simple. The rules outline the *BrowsePaths* of the properties of the *Types*, the instances of which are to be compared across the models to decide whether the instances represent the same entity i.e. the Pump in our example scenario e.g there is just one rule in the JSON containing the IMR between the DEXPI and PADIM Models (6.1) and two rules corresponding to the two pairs of properties that need to be compared in the JSON containing the IMR between the PADIM and OPC 40223 models (6.2). During the aggregation process these rules are loaded by the Aggregation Node Manager to merge the instances within the *AggregatedPumpType* instance in the address space of the Aggregation Server. These rules were kept in a local storage location.

```
{
  "model1Namespace" : "http://opcfoundation.org/UA/DEXPI/",
  "model2Namespace" : "http://opcfoundation.org/UA/PADIM/",
  "rules" :
  [
    {
      "model1TypeName" : "CentrifugalPumpType",
      "model2TypeName" : "PADIMType",
      "valuesToBeCompared" :
      [
        {
        "browsePath1" : "./TagNameAssignmentClass",
        "browsePath2" : "./SignalSet/SignalS1/SignalTag"
        }
      ]
    }
  ]
}
```

Listing 6.1: Instance Mapping Rule JSON for the Pump instance in DEXPI and PADIM models

```
{
  "model1Namespace" : "http://opcfoundation.org/UA/PADIM/",
  "model2Namespace" : "http://opcfoundation.org/UA/Pumps/",
  "rules" :
  [
  {
    "model1TypeName" : "PADIMType",
    "model2TypeName" : "PumpType",
    "valuesToBeCompared" :
    [
      {
        "browsePath1" : "./Manufacturer",
        "browsePath2" : "./Identification/Manufacturer"
      },
      {
        "browsePath1" : "./SerialNumber",
        "browsePath2" : "./Identification/SerialNumber"
      }
    ]
  }
  ]
}
```

Listing 6.2: Instance Mapping Rule JSON for the Pump instance in PADIM and OPC 40223 models

Auto-creation of rules

As mentioned in 5.2.1, the pre-requisite for the auto-creation of new mapping rules is the existence of a transitive relationship between the existing rules with respect to the parameters being compared among the models. Unfortunately in our example scenario, a transitive relationship does not exist. So in order to test the functioning of the automatic creation of rules, an IMR between the PADIM Model and a fictitious ModelX was manually created and added to the local mapping repository (listing 6.3). In this fictitious ModelX the Manufacturer and the SerialNumber can be found within a so-called *AssetInfo* object within the instance representing the pump in the address space. A Python script was programmed to monitor the addition of any new mapping rules to the local repository. As soon as a new mapping rule would be added, it would trigger the algorithm that looks for the transitive relationship and creates a new mapping rule between the two appropriate models e.g. in the test case mentioned above, a new IMR was created by the script between the fictitious ModelX and the OPC 40223 models (listing 6.4) and the properties of Pump instances to be compared between ModelX and OPC 40223 models were populated correctly.

```
  {
    "model1Namespace" : "http://opcfoundation.org/UA/PADIM/",
    "model2Namespace" : "http://fictitiousmodel.org/PumpX/",
    "rules" :
    [
    {
      "model1TypeName" : "PADIMType",
      "model2TypeName" : "PumpXType",
      "valuesToBeCompared" :
      [
      {
        "browsePath1" : "./Manufacturer",
        "browsePath2" : "./AssetInfo/ManufacturerName"
      },
      {
        "browsePath1" : "./SerialNumber",
        "browsePath2" : "./AssetInfo/SNo"
      }
      ]
    }

    ]
  }
```

Listing 6.3: Instance Mapping Rule JSON for the Pump instance in PADIM and ModelX models

```
  {
    "model1Namespace" : "http://opcfoundation.org/UA/Pumps/",
    "model2Namespace" : "http://fictitiousmodel.org/PumpX/",
    "rules" :
    [
    {
      "model1TypeName" : "PumpType",
      "model2TypeName" : "PumpXType",
      "valuesToBeCompared" :
      [
      {
        "browsePath1" : "./Identification/Manufacturer",
        "browsePath2" : "./AssetInfo/ManufacturerName"
      },
      {
        "browsePath1" : "./Identification/SerialNumber",
        "browsePath2" : "./AssetInfo/SNo"
      }
      ]
    }

    ]
  }
```

Listing 6.4: Instance Mapping Rule JSON for the Pump instance in PADIM and ModelX models

6.4 Evaluation of the implemented prototype

Inline with the V-Model from 3.1.1 the implemented prototype was validated with respect to the behavioural model described in section 5.4 in the previous chapter. The three underlying OPC UA Servers (PADIM, DEXPI and OPC 40223) were aggregated by the Aggregation Server. The entire UIMs within the underlying servers were represented by the AIM hosted by the Aggregation Server. The three Pump instances in the underlying servers (Pump_DEXPI, Pump_PADIM and Pump_40223) were aggregated (along with their respective parameters) correctly within an *AggregatedPump* instance in the Aggregation Server. The data access services (*Read, Write, Subscribe*) were tested and validated. Furthermore, the implementation was validated against the functional and non-functional requirements mentioned in 4.2.2. Table 6.1 and Table 6.2 provide an overview of the validation of the prototype implementation against the requirements. Annex A provides screenshots of the UAExpert Client (AIC) showing the aggregation result in the GUI of the client and tries to provide an overview of the results while testing the implementation to test the functioning of the *Read, Write and Subscribe* services (see Figures A.4,A.5,A.6,A.7,A.8).

Requirement	**Description**
User Roles	fulfilled by OPC UA User Management
Completeness	fulfilled by aggregating all elements in the underlying server address space
Handling Underlying Models	fulfilled by the inclusion of dynamic aggregation
Information Access	fulfilled by the OPC UA Browse and Read Services
Function Support	fulfilled by extension of the working principle used for Read and Write
Read/Write Data	fulfilled see Figure A.6 and Figure A.7
Subscriptions	fulfilled see Figure A.8
Online/Offline server will too	if the underlying server supports this, the aggregation
Alarms	fulfilled by extension of the working principle
Handle Mapping Rulesets	fulfilled by the Mapping Respository
Semantic	fulfilled by the usage of *AggregatedInstanceType*

Table 6.1: Functional requirements fulfilled by the solution concept

Requirement	**Description**
Platform-Independence	fulfilled by the usage of JSON, Python and OPC UA
Robust	fulfilled by the usage of OPC UA and complete aggregation of underlying servers
Domain-Independence	fulfilled by aggregating Types and Instances irrespective of the domain
Openness	fulfilled by the usage of OPC UA
Transparent	fulfilled by design, clients do not need any communication specific knowledge except OPC UA
Multiclient	fulfilled, multiple OPC UA Clients can connect simultaneously
Multisource	fulfilled, multiple servers were successfully aggregated
Secure	fulfilled by the usage of OPC UA by design

Table 6.2: Non-functional requirements fulfilled by the solution concept

6.5 Summary

In this chapter, the realization and review of the Aggregated Integration Platform was discussed. Based on requirements and the meta-model from Chapter 4 and the software architecture from Chapter 5, the implementation technologies for the description of the mapping rules and for the client-server architecture were chosen. Section 6.3 discusses the implementation of each component of the platform itself as well as the underlying servers that were aggregated by the platform. In 6.3.5 the implementation of the mapping rules (needed for the aggregation process) and the mapping repository was discussed. The auto-creation of these rules was demonstrated in 6.3.5. Section 6.4 briefly discussed the validation of the implemented prototype.

Chapter 7

Conclusion and Outlook

This concluding chapter summarizes the work in this thesis and the result achieved in the process. Section 7.1 presents a walkthrough of the thesis starting from the problem definition,then the proposed solution and finally its prototype implementation. This is followed by an overall evaluation of the results and an outlook on future research activities that might follow this thesis (7.2).

7.1 Conclusion

With the advent of Industry 4.0, information modelling and information models have become ever more important in automation technology. Devices used in today's automation domain are becoming cheaper, smaller but at the same time more powerful. Due to this reason, they are now offering a lot more extensive data and functionality, thereby creating the need for efficient information models providing access to the data and functionality. These information models are constantly evolving and can be adapted to the needs of the industry so that they are best suited for a particular application or use-case e.g. predictive maintenance on the basis of condition monitoring parameters of a device, a subsystem of devices or the overall system itself. While the advantages of using efficient information models are manifold, it is also becoming increasingly important to identify commonalities between the information models and thereby the provision of a common landscape of information models. The activities that are involved in the management of such information models during the entire life cycle of an automation system, have been termed as Information Model-Management within the scope of this thesis. As the strict hierarchy between the different levels of the automation pyramid is now getting blurry, these information models are being used to provide access to different aspects of the same system from anywhere in the pyramid e.g. ERP systems are being able to access the Asset Management related information of a field device

directly, MES systems are being able to send Jobs to machines on the shop floor directly e.t.c and all this without going through the other layers of the pyramid. The natural consequence of this is the resulting connection mesh with a lot of information consumers being simultaneously connected to a multitude of information providers. This in turn makes the management of the information models difficult or manufacturer specific closed information models.
To solve this problem, this thesis aims to develop an Aggregated Integration Platform for the efficient management of information models. This platform can be used to aggregate several information models in an Aggregated Information Model which exposes data and functionality of all the information models being aggregated, in an efficient way. In order to do this, this thesis first analyses the methods and tools for information integration and looks at the information modeling process altogether with the aim to find a common landscape for information models. The thesis then investigates the different actors and the use-cases for which information models are being used in today's automation scenarios and thereafter the functional and non-functional requirements for the Aggregated Integration platform were defined. Upon reviewing several information models and an example scenario of overlapping information models, a concept to solving this overlap was discussed i.e. the creation of a merged information model. With this knowledge, a meta model for the platform was conceived which represents the Information Model-Management in three layers i.e. *Aggregated Information Model, Aggregation Communication Model and Consumer Operation Model.* With the meta-model as its basis, a software architecture for the platform was developed. The structural as well as the behavioural model of the architecture was defined based on the interactions between the three aforementioned layers. Several Information Models (named Underlying Information Models in this these) are aggregated into a single Aggregated Information Models, and there might be several Consumer Operation Models interacting with the Aggregated Information Model. In order to solve the overlaps between the Underlying Information Models, a concept of Instance and Type Mapping Rules and the management of these rules within a Mapping Repository was conceptualized within the software-architecture. This led to a prototype implementation following a Client-Server architecture, whereby the Underlying Information Models were hosted in their own respective Servers which were then aggregated by an Aggregation Server with the help of Aggregation Clients. As can be noticed from Table 6.1 and Table 6.2 in the previous chapter, the evaluation of the prototype against the expected behaviour of such an aggregated platform showed the developed concept does fulfil the requirements that had been set for it. The Aggregated Information Platform makes it possible for applications to be able to access the information about a particular entity all-in-one-place, thereby enabling use-case specific access to the aggregated information. Point to point con-

nections between applications to several information sources can then be omitted and the effort will be reduced accordingly.

7.2 Outlook

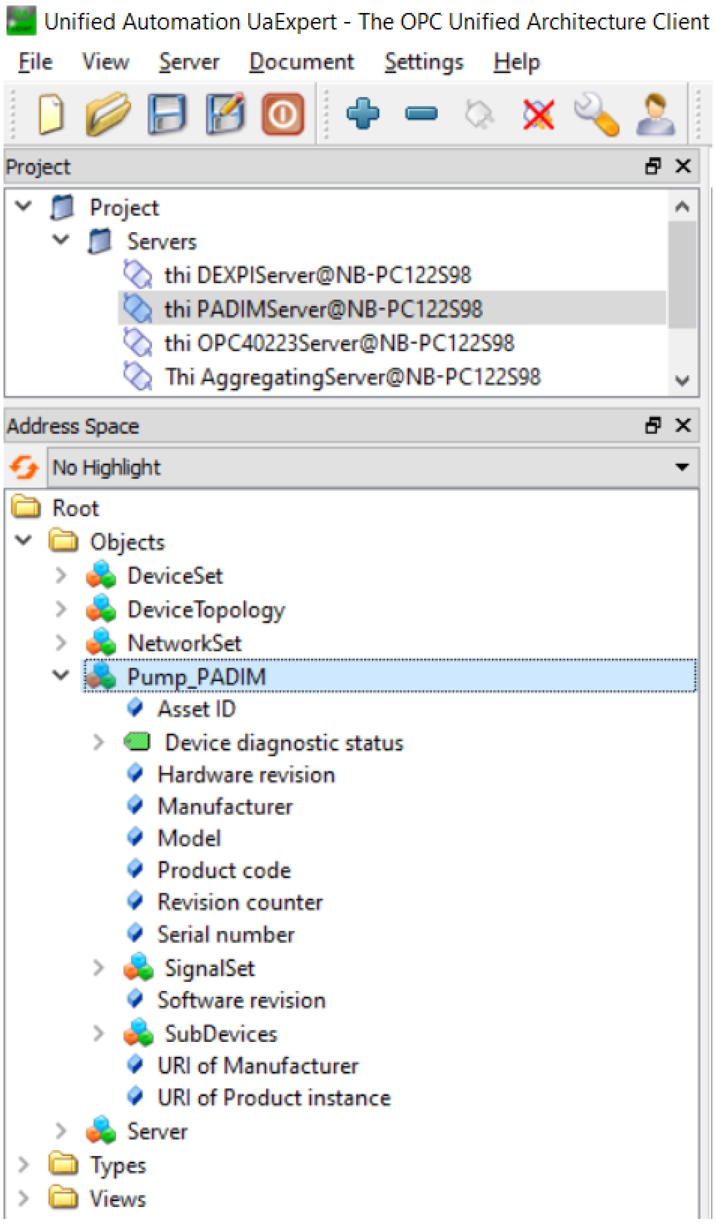
The concept developed within the scope of this thesis can be used to bring together information from different information sources that is essentially about different aspects of the same or similar entities. With the advent of Machine Learning in the automation industry, such aggregated information can help solve the prime use-cases of predictive and preventive maintenance e.g. if the aggregated information about the Pump mentioned in the example scenario in this thesis is monitored over a period of time, it might be possible to run pattern recognition algorithms on the dataset to be able to identify data patterns that were observed in relation to a reported malfunction. The next time a similar data pattern under similar conditions is observed during the lifetime of the Pump, another malfunction might be predicted in the same time frame as observed the previous times this had happened. Needless to say that the same concept can be extended to any asset which is prone to malfunction as some point in time (e.g. wear and tear of a physical asset, existence of bugs in a software asset). As with any other Machine Learning system, the efficiency of such a prediction will improve with time as the system learns more and more. Since such an aggregated platform is capable of bringing correlated information, the dataset would not need to be collected from many different sources, rather it will be available all-in-one-place. Since the correlation between the information obtained from different sources can be more easily observed in an aggregated model, it will also make it easier to identify the causation of observations. An example of this can be the currently discussed topic of energy efficiency and optimisations whereby all the factors resulting in a particular value of energy consumption might not be easily apparent. Observation of the parameters in an aggregated platform may help in identifying the data points that might play role in making a particular device/sub-system more energy efficient. Moreover, the aggregation process itself can be use-case specific i.e. instead of aggregating everything from underlying sources only information related to energy consumption/efficiency can be aggregated. Such a use-case specific aggregation can be done at the subsystem level (which together build up an entire system). Several such subsystems can further be aggregated to provide a system level use-case specific aggregated information model.

Appendices

Annex A

Criteria	**JSON**	**XML**
Platform-Independence	+	+
Ease of Use	+	-
Human-readable	+	+
Hierarchical	+	+
Speed of use	+	-
Secure	+	-

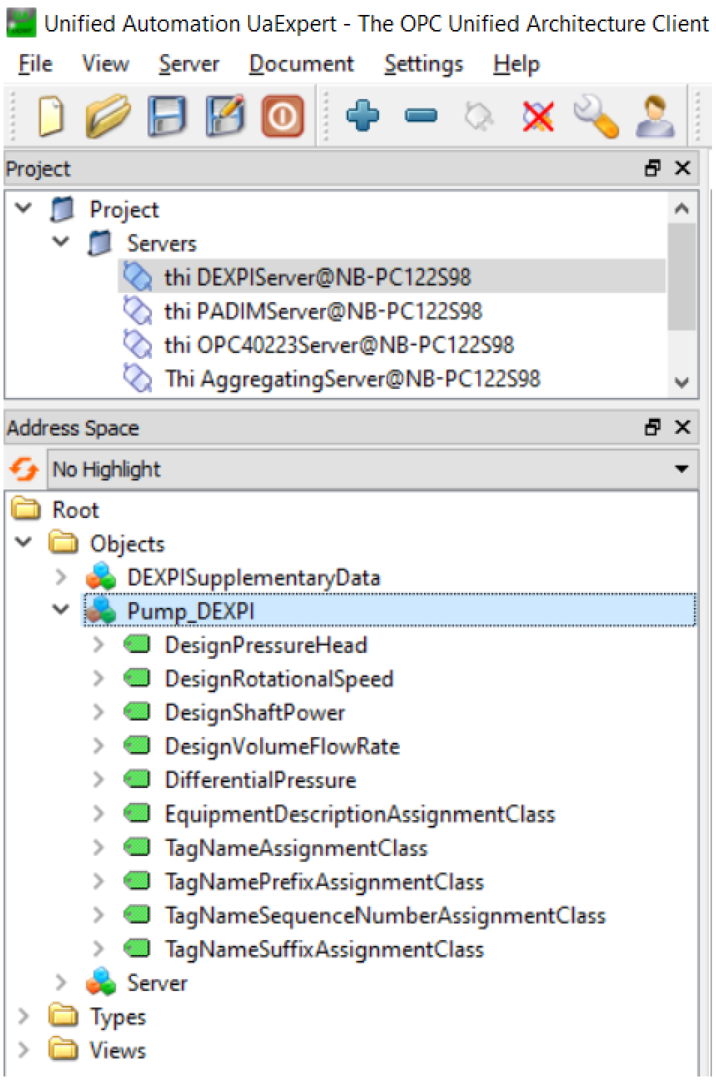
Table A.1: Comparison of JSON and XML



```
Project
  Servers
    thi DEXPIServer@NB-PC122S98
    thi PADIMServer@NB-PC122S98
    thi OPC40223Server@NB-PC122S98
    Thi AggregatingServer@NB-PC122S98
```

```
Root
Objects
  DeviceSet
  DeviceTopology
  NetworkSet
  Pump_PADIM
    Asset ID
    Device diagnostic status
    Hardware revision
    Manufacturer
    Model
    Product code
    Revision counter
    Serial number
    SignalSet
    Software revision
    SubDevices
    URI of Manufacturer
    URI of Product instance
  Server
Types
Views
```

Figure A.1: Pump instance in the PADIM OPC UA Server Address Space



Unified Automation UaExpert - The OPC Unified Architecture Client

Project

- Project
  - Servers
    - thi DEXPIServer@NB-PC122S98
    - thi PADIMServer@NB-PC122S98
    - thi OPC40223Server@NB-PC122S98
    - Thi AggregatingServer@NB-PC122S98

Address Space

No Highlight

- Root
  - Objects
    - DEXPISupplementaryData
    - Pump_DEXPI
      - DesignPressureHead
      - DesignRotationalSpeed
      - DesignShaftPower
      - DesignVolumeFlowRate
      - DifferentialPressure
      - EquipmentDescriptionAssignmentClass
      - TagNameAssignmentClass
      - TagNamePrefixAssignmentClass
      - TagNameSequenceNumberAssignmentClass
      - TagNameSuffixAssignmentClass
    - Server
  - Types
  - Views

Figure A.2: Pump instance in the DEXPI OPC UA Server Address Space

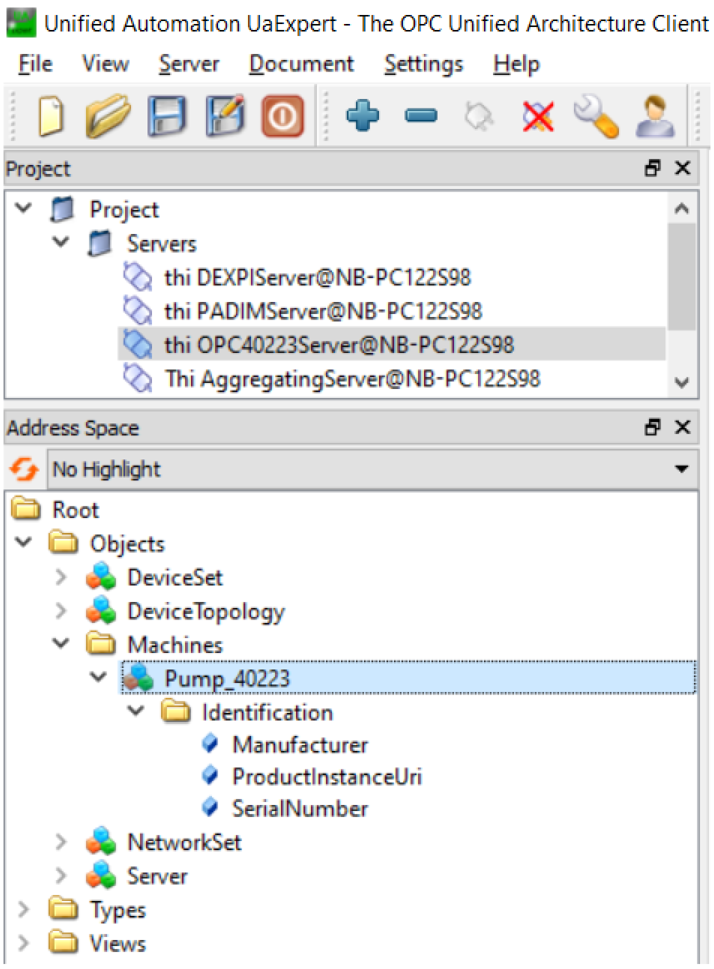


Figure A.3: Pump instance in the OPC 40223 OPC UA Server Address Space

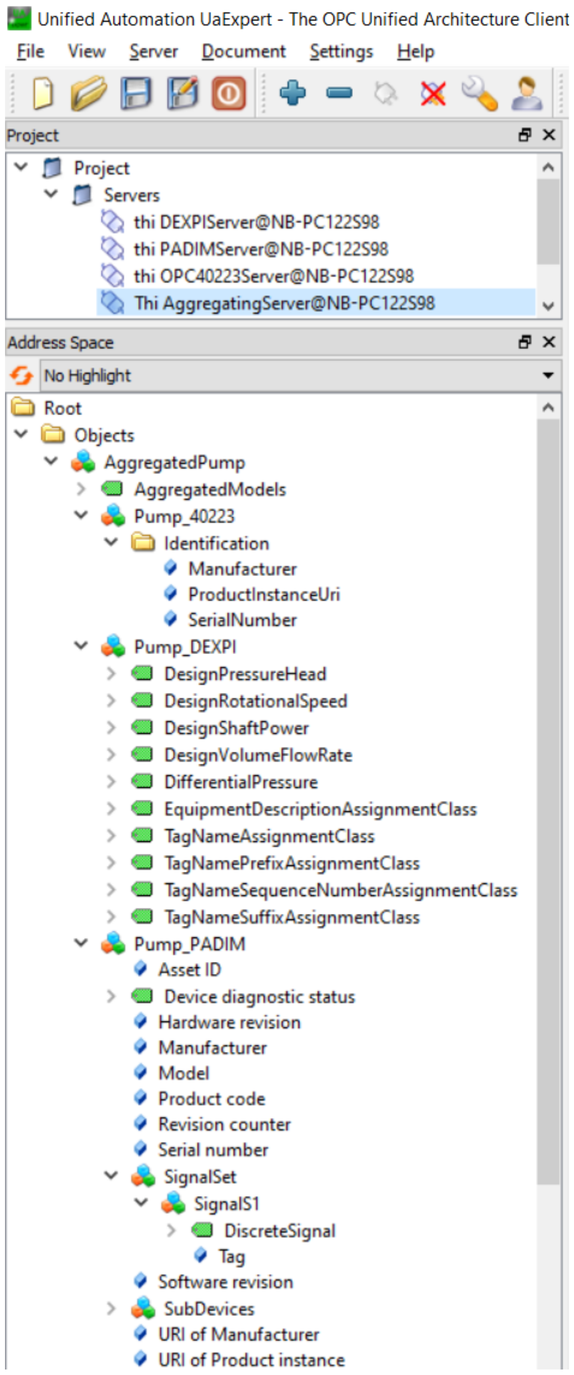
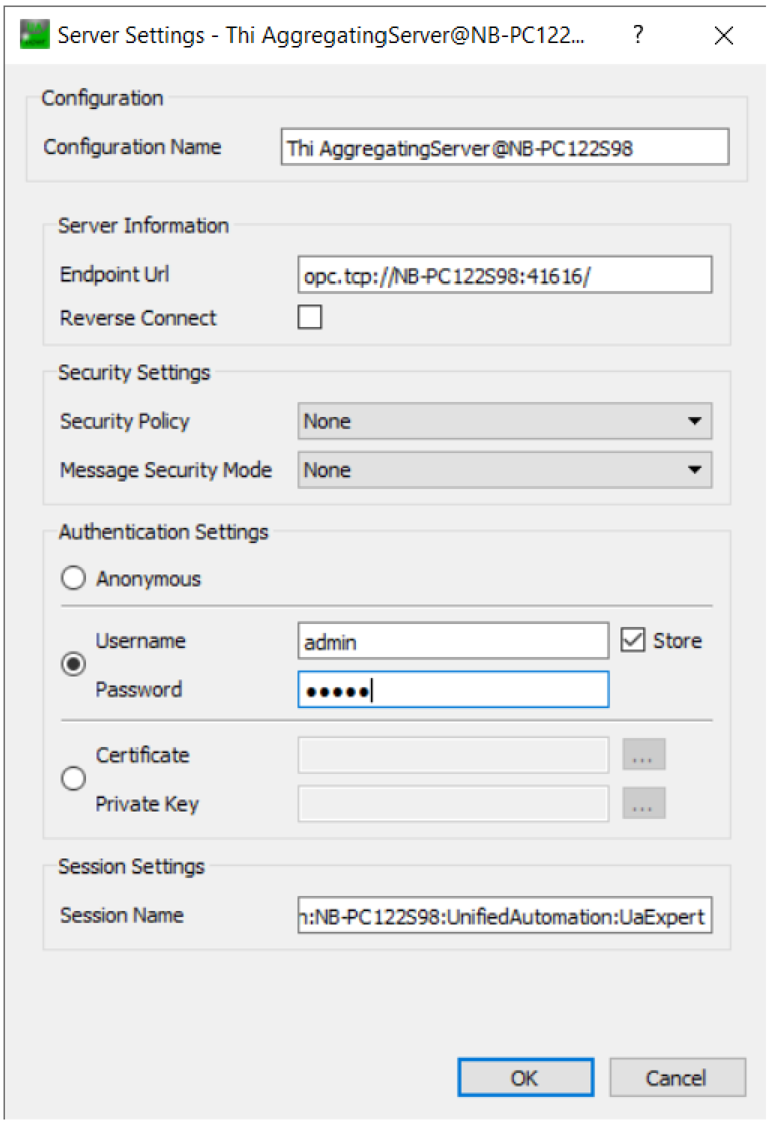


Figure A.4: AggregatedPump instance in the Aggregation OPC UA Server Address Space


Server Settings - Thi AggregatingServer@NB-PC122...

Configuration

Configuration Name	Thi AggregatingServer@NB-PC122S98

Server Information

Endpoint Url	opc.tcp://NB-PC122S98:41616/
Reverse Connect	☐

Security Settings

Security Policy	None
Message Security Mode	None

Authentication Settings

○ Anonymous

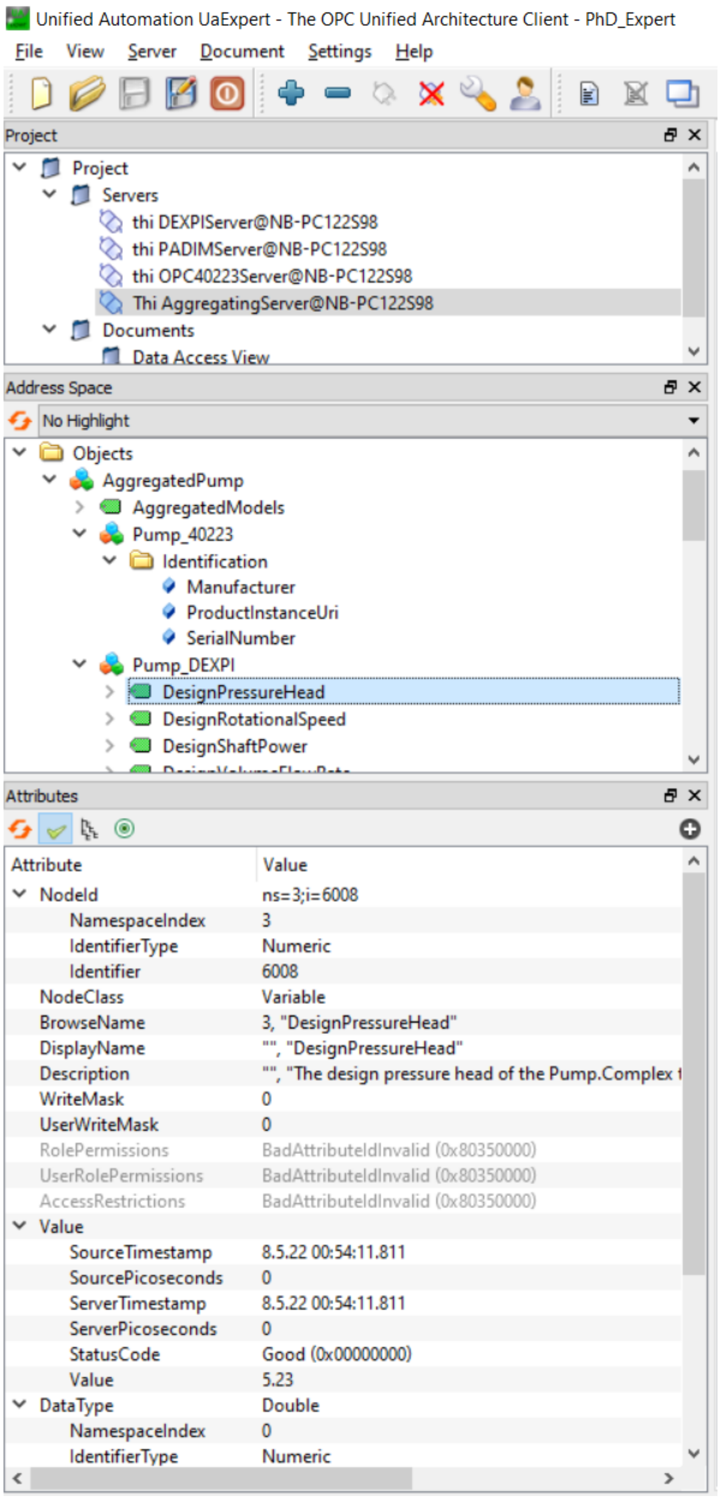
● Username: admin ☑ Store
Password: •••••

○ Certificate
Private Key

Session Settings

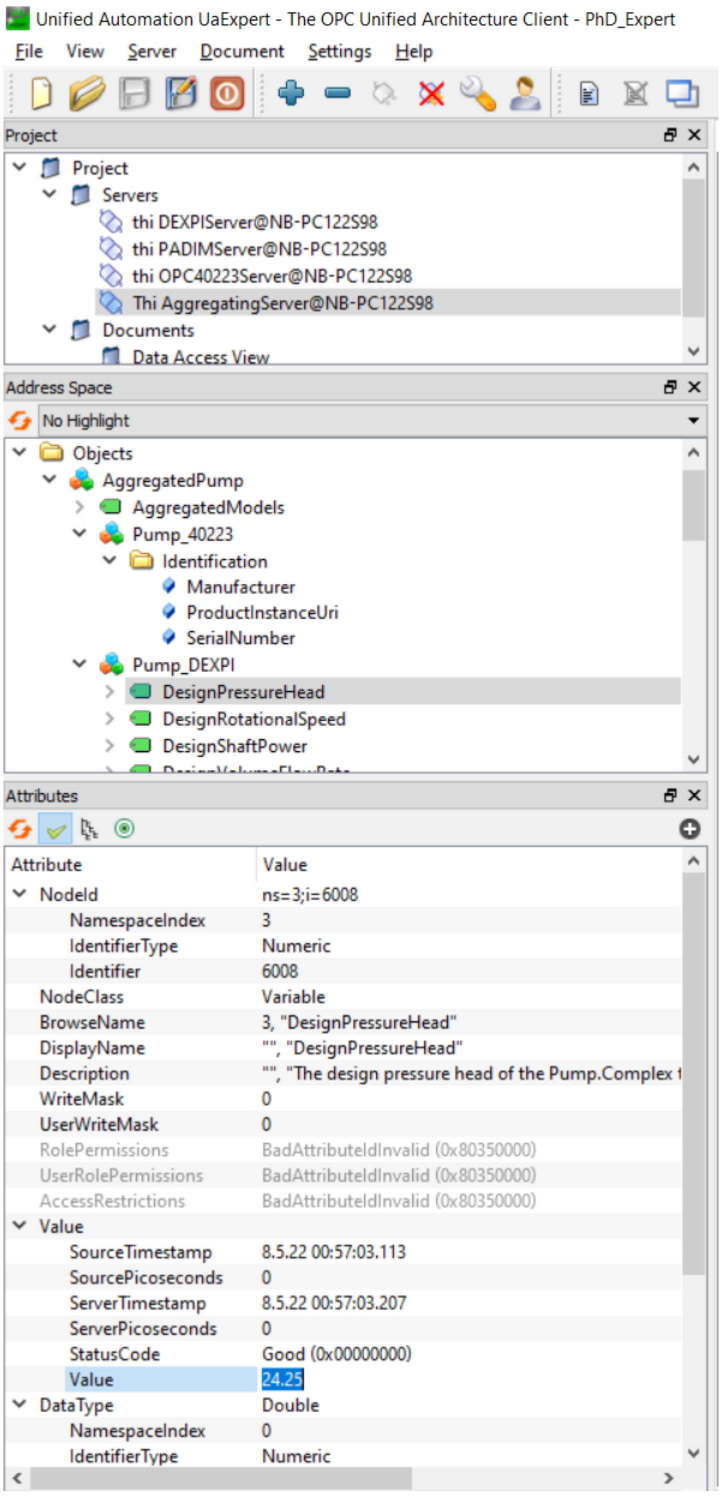
Session Name	n:NB-PC122S98:UnifiedAutomation:UaExpert

Figure A.5: Login Service in the Aggregation Server



Attribute	Value
NodeId	ns=3;i=6008
NamespaceIndex	3
IdentifierType	Numeric
Identifier	6008
NodeClass	Variable
BrowseName	3, "DesignPressureHead"
DisplayName	"", "DesignPressureHead"
Description	"", "The design pressure head of the Pump.Complex
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeIdInvalid (0x80350000)
UserRolePermissions	BadAttributeIdInvalid (0x80350000)
AccessRestrictions	BadAttributeIdInvalid (0x80350000)
Value	
SourceTimestamp	8.5.22 00:54:11.811
SourcePicoseconds	0
ServerTimestamp	8.5.22 00:54:11.811
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	5.23
DataType	Double
NamespaceIndex	0
IdentifierType	Numeric

Figure A.6: Read Service in the Aggregation Server

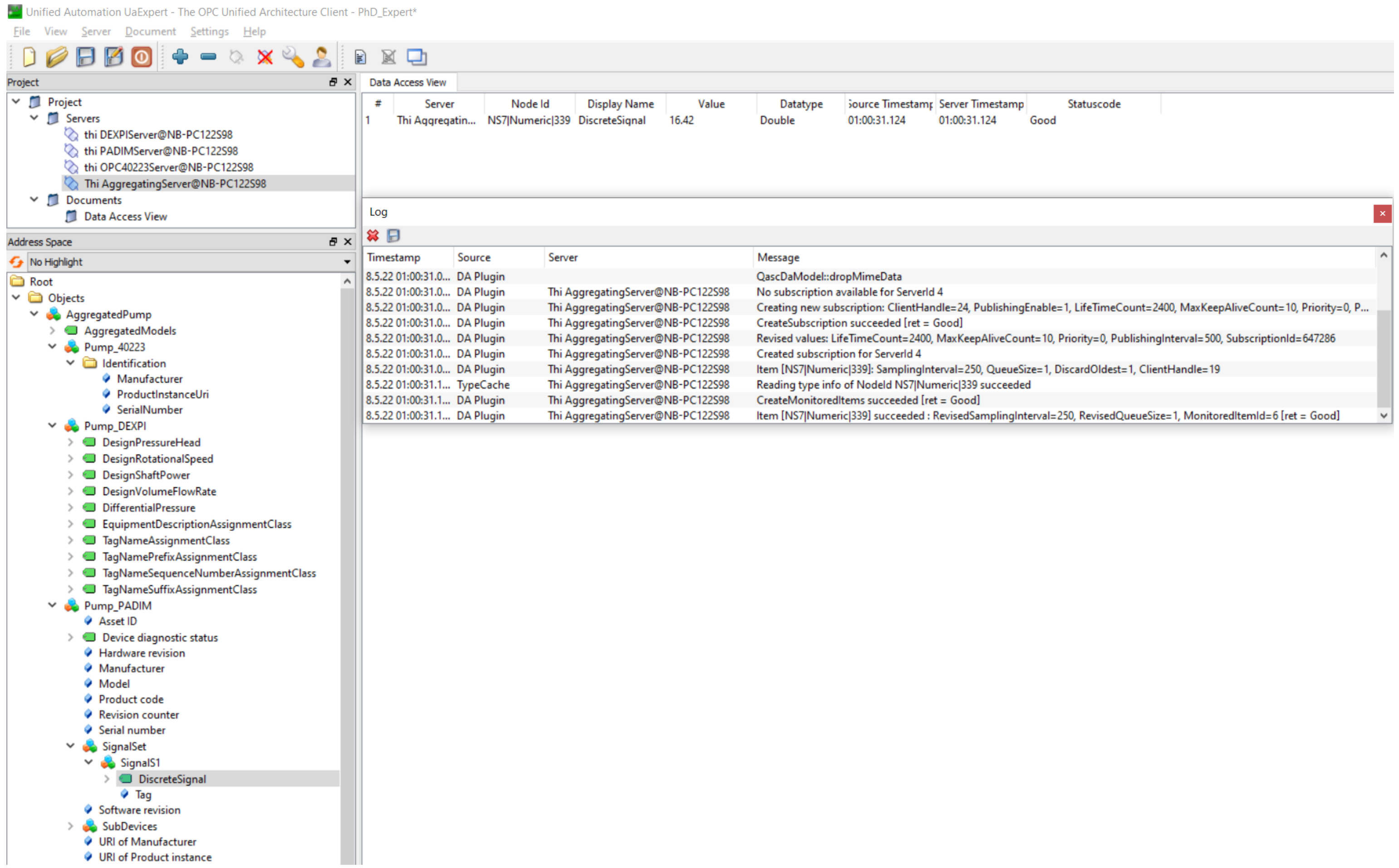

Unified Automation UaExpert - The OPC Unified Architecture Client - PhD_Expert

- Project
  - Servers
    - thi DEXPIServer@NB-PC122S98
    - thi PADIMServer@NB-PC122S98
    - thi OPC40223Server@NB-PC122S98
    - Thi AggregatingServer@NB-PC122S98
  - Documents
    - Data Access View

- Objects
  - AggregatedPump
    - AggregatedModels
    - Pump_40223
      - Identification
        - Manufacturer
        - ProductInstanceUri
        - SerialNumber
    - Pump_DEXPI
      - DesignPressureHead
      - DesignRotationalSpeed
      - DesignShaftPower

Attribute	Value
NodeId	ns=3;i=6008
NamespaceIndex	3
IdentifierType	Numeric
Identifier	6008
NodeClass	Variable
BrowseName	3, "DesignPressureHead"
DisplayName	"", "DesignPressureHead"
Description	"", "The design pressure head of the Pump.Complex
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeIdInvalid (0x80350000)
UserRolePermissions	BadAttributeIdInvalid (0x80350000)
AccessRestrictions	BadAttributeIdInvalid (0x80350000)
Value	
SourceTimestamp	8.5.22 00:57:03.113
SourcePicoseconds	0
ServerTimestamp	8.5.22 00:57:03.207
ServerPicoseconds	0
StatusCode	Good (0x00000000)
Value	24.25
DataType	Double
NamespaceIndex	0
IdentifierType	Numeric

Figure A.7: Write Service in the Aggregation Server



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	Thi Aggregatin...	NS7|Numeric|339	DiscreteSignal	16.42	Double	01:00:31.124	01:00:31.124	Good

```
Timestamp           Source      Server                              Message
8.5.22 01:00:31.0...  DA Plugin                                     QascDaModel::dropMimeData
8.5.22 01:00:31.0...  DA Plugin   Thi AggregatingServer@NB-PC122S98   No subscription available for ServerId 4
8.5.22 01:00:31.0...  DA Plugin   Thi AggregatingServer@NB-PC122S98   Creating new subscription: ClientHandle=24, PublishingEnable=1, LifeTimeCount=2400, MaxKeepAliveCount=10, Priority=0, P...
8.5.22 01:00:31.0...  DA Plugin   Thi AggregatingServer@NB-PC122S98   CreateSubscription succeeded [ret = Good]
8.5.22 01:00:31.0...  DA Plugin   Thi AggregatingServer@NB-PC122S98   Revised values: LifeTimeCount=2400, MaxKeepAliveCount=10, Priority=0, PublishingInterval=500, SubscriptionId=647286
8.5.22 01:00:31.0...  DA Plugin   Thi AggregatingServer@NB-PC122S98   Created subscription for ServerId 4
8.5.22 01:00:31.0...  DA Plugin   Thi AggregatingServer@NB-PC122S98   Item [NS7|Numeric|339]: SamplingInterval=250, QueueSize=1, DiscardOldest=1, ClientHandle=19
8.5.22 01:00:31.1...  TypeCache   Thi AggregatingServer@NB-PC122S98   Reading type info of NodeId NS7|Numeric|339 succeeded
8.5.22 01:00:31.1...  DA Plugin   Thi AggregatingServer@NB-PC122S98   CreateMonitoredItems succeeded [ret = Good]
8.5.22 01:00:31.1...  DA Plugin   Thi AggregatingServer@NB-PC122S98   Item [NS7|Numeric|339] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=6 [ret = Good]
```

Figure A.8: Subscribe Service in the Aggregation Server

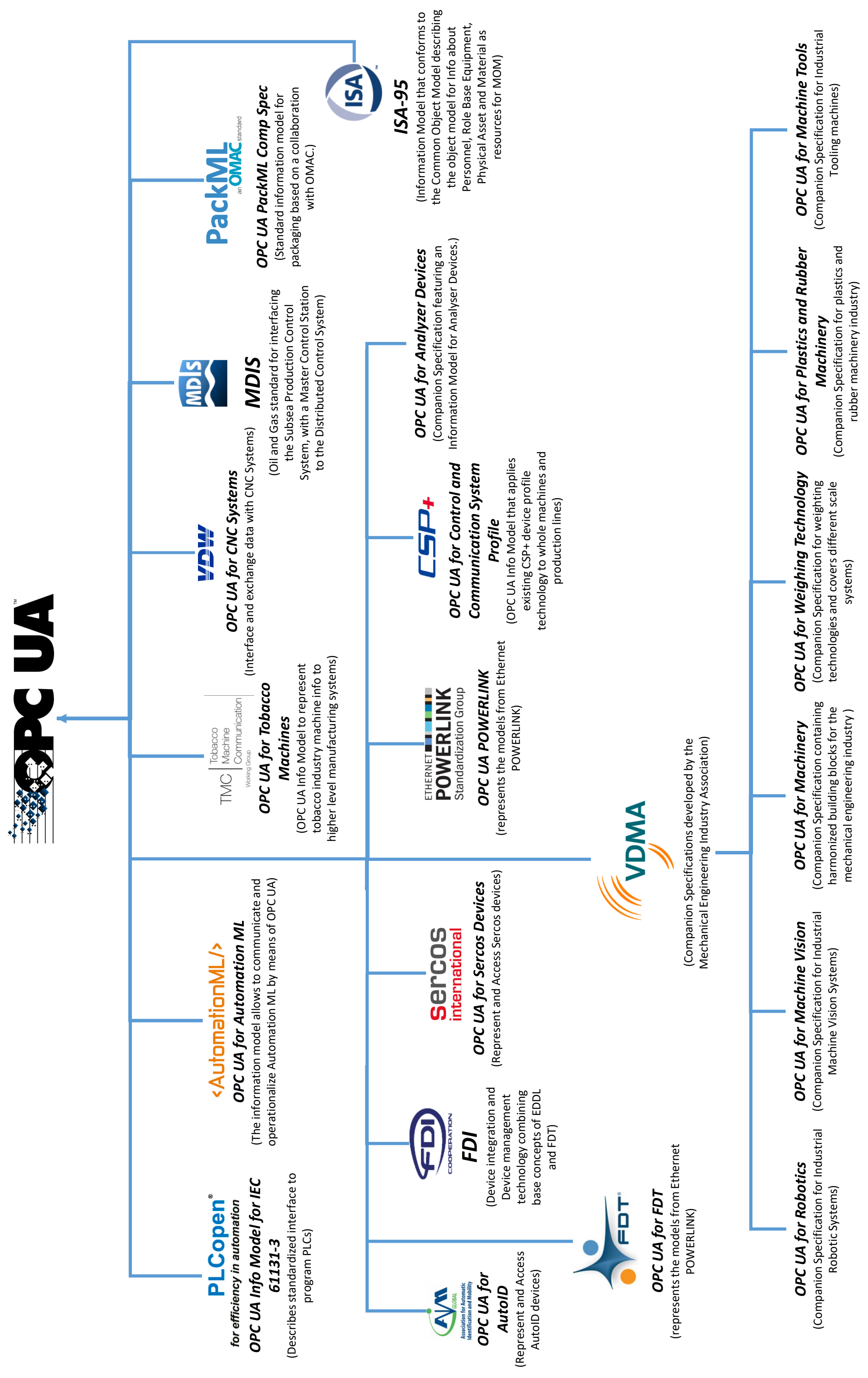

OPC UA
PLCopen for efficiency in automation
OPC UA Info Model for IEC 61131-3
(Describes standardized interface to program PLCs)
<AutomationML/>
OPC UA for Automation ML
(The information model allows to communicate and operationalize Automation ML by means of OPC UA)
TMC Tobacco Machine Communication Working Group
OPC UA for Tobacco Machines
(OPC UA Info Model to represent tobacco industry machine info to higher level manufacturing systems)
VDW
OPC UA for CNC Systems
(Interface and exchange data with CNC Systems)
MDIS
MDIS
(Oil and Gas standard for interfacing the Subsea Production Control System, with a Master Control Station to the Distributed Control System)
PackML an OMAC standard
OPC UA PackML Comp Spec
(Standard information model for packaging based on a collaboration with OMAC)
ISA
ISA-95
(Information Model that conforms to the Common Object Model describing the object model for Info about Personnel, Role Base Equipment, Physical Asset and Material as resources for MOM)
AIM GLOBAL Association for Automatic Identification and Mobility
OPC UA for AutoID
(Represent and Access AutoID devices)
FDI COOPERATION
FDI
(Device integration and Device management technology combining base concepts of EDDL and FDT)
sercos international
OPC UA for Sercos Devices
(Represent and Access Sercos devices)
ETHERNET POWERLINK Standardization Group
OPC UA POWERLINK
(represents the models from Ethernet POWERLINK)
CSP+
OPC UA for Control and Communication System Profile
(OPC UA Info Model that applies existing CSP+ device profile technology to whole machines and production lines)
OPC UA for Analyzer Devices
(Companion Specification featuring an Information Model for Analyser Devices.)
FDT
OPC UA for FDT
(represents the models from Ethernet POWERLINK)
VDMA
(Companion Specifications developed by the Mechanical Engineering Industry Association)
OPC UA for Robotics
(Companion Specification for Industrial Robotic Systems)
OPC UA for Machine Vision
(Companion Specification for Industrial Machine Vision Systems)
OPC UA for Machinery
(Companion Specification containing harmonized building blocks for the mechanical engineering industry)
OPC UA for Weighing Technology
(Companion Specification for weighting technologies and covers different scale systems)
OPC UA for Plastics and Rubber Machinery
(Companion Specification for plastics and rubber machinery industry)
OPC UA for Machine Tools
(Companion Specification for Industrial Tooling machines)

Figure A.9: OPC UA Companion Standards [MLD09] [Fou17]

List of Figures

1.1 The gradual phase-in of the automation pyramid 2
1.2 Structure of the Thesis 4

2.1 The Automation Pyramid - Hierarchy of Automation Systems . . . 10
2.2 Phase plan as per [NAM19] 11
2.3 Phase plan of the Service phase as per [FMS04] 12
2.4 The IoT Device Lifecycle Management Model (adapted from [Soo+18]) 13
2.5 Structure of Cyber Physical Production Systems 16
2.6 Automation Pyramid in a CPPS perspective 18
2.7 Resulting Communication Mesh 19
2.8 Enterprise Application Integration 20
2.9 The Aggregation Layer 22

3.1 The Information Integration Process (adapted from [VDI21]) 29
3.2 Identification of Communication Interfaces 30
3.3 Main components of an information model 32
3.4 The Four Worlds of Information Systems Engineering (as defined by the DAIDA project [Jar+92]) 34
3.5 An example of a simple Graph Information Model 36
3.6 Information Model Development Steps 38
3.7 Important Organizations and Standardization contributions 44
3.8 Reference Architecture Model Industrie 4.0 45
3.9 OPC UA Layered Architecture 48
3.10 The OPC UA Client-Server Relationships 50
3.11 OPC UA - An Extensible Model 52

4.1 Basic Concept of the Aggregation Platform 57
4.2 Procedure for the meta-model development 59
4.3 Actors 60
4.4 Use-Cases 62
4.5 The Digital-Representation Concept 67

4.6 An Aggregation Scenario from the Process Industry 70
4.7 Instance and Type Mapping Rules 75
4.8 Instance and Type Mapping Rules 77
4.9 Activity Diagram for the aggregation of types and instances 79
4.10 DEXPI Graph Information Model of the Pump 83
4.11 PA-DIM Graph Information Model of the Pump 85
4.12 OPC 40223 Graph Information Model of the Pump 87
4.13 Merged Graph Information Model of the Pump 91
4.14 The Base Model 93
4.15 Aggregated Information Model 94
4.16 Aggregation Communication Model 96
4.17 Consumer Operation Model 98
4.18 Modeling Levels of MOF [OMG06] 100
4.19 Basic Structure of the Meta-model 101
4.20 Data and State Model 103
4.21 Basic Aggregation Methods 104
4.22 Addressing Information 105
4.23 Communication Sequences 106
4.24 Consumer Business Logic 106
4.25 Graphical User Interface 107

5.1 The Architecture Concept 112
5.2 Procedure for the Architecture Development 113
5.3 Login 115
5.4 AIM 117
5.5 COM 118
5.6 ACM 119
5.7 Configuration 119
5.8 Automatic Rule Creation (based on a transitive relationship) 121
5.9 Mapping 121
5.10 Deployment Diagram for Mapping Repositories 122
5.11 The Interfaces 123
5.12 Getting the AIM element Id 128
5.13 Reading a UIM element 129
5.14 Writing to a UIM element 130
5.15 Subscribing to a UIM element 131
5.16 Subscription Data Change Event 132
5.17 Addition of an element by the consumer 133
5.18 Deletion of an element by the Consumer 134
5.19 Dynamic Addition of a UIM element 136
5.20 Dynamic Deletion of a UIM element 137

6.1 Overview of the prototype 141
6.2 Process of Prototype Implementation 142
6.3 Implemented modules in the aggregated platform 144
6.4 Pump instance in the PADIM OPC UA Server 145
6.5 Pump instance in the DEXPI OPC UA Server 146
6.6 Pump instance in the OPC 40223 OPC UA Server 147
6.7 Aggregated Pump instance in the Aggregation OPC UA Server . . 149

A.1 Pump instance in the PADIM OPC UA Server Address Space . . . 164
A.2 Pump instance in the DEXPI OPC UA Server Address Space . . . 165
A.3 Pump instance in the OPC 40223 OPC UA Server Address Space . 166
A.4 AggregatedPump instance in the Aggregation OPC UA Server Address Space 167
A.5 Login Service in the Aggregation Server 168
A.6 Read Service in the Aggregation Server 169
A.7 Write Service in the Aggregation Server 170
A.8 Subscribe Service in the Aggregation Server 171
A.9 OPC UA Companion Standards 172

List of Tables

4.1 Description of Actors 61
4.2 Description of the Use-Cases 63
4.3 Functional Requirements 64
4.4 Non-functional Requirements 65
4.5 Aggregation Scenarios 71

6.1 Functional requirements fulfilled by the solution concept 155
6.2 Non-functional requirements fulfilled by the solution concept 156

A.1 Comparison of JSON and XML 163

Listings

4.1 RDF Dataset for the DEXPI example graph 82
4.2 RDF Dataset for the PADIM example graph 84
4.3 RDF Dataset for the OPC40223 example graph 86
4.4 SPARQL query to match the two pump instances between the DEXPI and PA-DIM models 88
4.5 SPARQL query to match the two pump instances between the PA-DIM and OPC 40223 models 89
4.6 RDF Dataset for the Merged graph 92
6.1 Instance Mapping Rule JSON for the Pump instance in DEXPI and PADIM models 152
6.2 Instance Mapping Rule JSON for the Pump instance in PADIM and OPC 40223 models 152
6.3 Instance Mapping Rule JSON for the Pump instance in PADIM and ModelX models 153
6.4 Instance Mapping Rule JSON for the Pump instance in PADIM and ModelX models 154

Acronyms

AAS	Asset Administration Shell. 70
ACM	Aggregation Communication Model. 95, 100
AI	Addressing Information. 96
AIM	Aggregated Information Model. 56, 63, 93
BM	Basic Methods. 95
BPMN	Business Process Model and Notation. 12
CBL	Consumer Business Logic. 98
COM	Consumer Operation Model. 97
CS	Communication Sequences. 97
DSM	Data and State Model. 94
EAI	Enterprise Application Integration. 20
ER	Entity-Relationship. 35
ERP	Enterprise Resource Planning. 11
GUI	Graphical User Interface. 99
HMI	Human Machine Interaction. 10
MES	Manufacturing Execution System. 10
MOF	Meta Object Facility. 99
OMG	Object Management Group. 99

O-O	Object-Oriented. 35, 36, 37
OPC UA	Open Platform Communication Unified Architecture. 46
PLC	Programmable Logic Controller. 10
SCADA	Supervisory Control and Data Acquisition. 10
SDK	Software Development Kit. 17
UML	Unified Modelling Language. 99

References

[Ado+16] P Adolphs, S Auer, H Bedenbender, M Billmann, M Hankel, R Heidel, M Hoffmeister, H Huhle, M Jochem, M Kiele-Dunsche, et al. 'Struktur der verwaltungsschale: Fortentwicklung des referenzmodells für die Industrie 4.0-komponente'. In: *Bundesministerium für Wirtschaft und Energie (BMW), Berlin* (2016), pp. 345–361.

[Aie06] S. Aier. *Enterprise application integration: Serviceorientierung und nachhaltige Architekturen.* Reihe Enterprise architecture. Gito-Verlag, 2006. ISBN: 9783936771749.

[ATE19] Selma Azaiez, Francois Tanguy, and Marc Engel. 'Towards building OPC-UA companions for semi-conductor domain'. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).* IEEE. 2019, pp. 142–149.

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* SEI series in software engineering. Addison-Wesley, 2003. ISBN: 9780321154958.

[BD10] B. Bruegge and A.H. Dutoit. *Object-oriented Software Engineering: Using UML, Patterns, and Java.* Prentice Hall, 2010. ISBN: 9780136061250.

[Bir01] Rolf Birkhofer. *Modellbasierte Beschreibung zur offenen Integration intelligenter Feldgeräte der Automatisierungstechnik.* Vol. 16. Herbert Utz Verlag, 2001.

[Bou+21] Cornelis Bouter, Monireh Pourjafarian, Leon Simar, and Robert Wilterdink. 'Towards a Comprehensive Methodology for Modelling Submodels in the Industry 4.0 Asset Administration Shell'. In: *2021 IEEE 23rd Conference on Business Informatics (CBI).* Vol. 2. IEEE. 2021, pp. 10–19.

[BPV12] Jörg Becker, Wolfgang Probandt, and Oliver Vering. *Grundsätze ordnungsmäßiger Modellierung: Konzeption und Praxisbeispiel für ein effizientes Prozessmanagement.* Springer-Verlag, 2012.

[Bus] F. Buschmann. *PatternOriented-Software-Architecture-A-System-of-Patterns-Volume-1*. Bd. 1. Wiley. URL: `https://books.google.de/books?id=OkUFZDuqvmEC`.

[Cdd] *IEC 61987 Common Data Dictionary*. Standard. International Electrotechnical Commission. URL: `https://cdd.iec.ch/cdd/iec61987/cdddev.nsf/TreeFrameset?OpenFrameSet`.

[Cle+10] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. SEI Series in Software Engineering. Pearson Education, 2010. ISBN: 9780132488594.

[Col+14] A.W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J.L. Lastra. *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*. SpringerLink : Bücher. Springer International Publishing, 2014. ISBN: 9783319056241.

[Com+13] International Electrotechnical Commission et al. 'IEC 62264-1: 2013'. In: *Enterprise-control system integration* (2013).

[Com16] International Electrotechnical Commission. *IEC TR 63022 ED1: Model Universals – Specification and Examples*. Sept. 2016.

[CS88] Allan Collins and Edward E Smith. 'Readings in Cognitive Science, a Perspective From Psychology and Artificial Intelligence'. In: (1988).

[DB06] C. Diedrich and T. Bangemann. *Profibus PA: Instrumentierungstechnologie für die Verfahrenstechnik*. Automatisierungstechnik 2016. Oldenbourg Industrieverl., 2006. ISBN: 9783835630567.

[DC+19] Jan De Caigny, Thomas Tauchnitz, Ronny Becker, Christian Diedrich, Tizian Schröder, Daniel Großmann, Suprateek Banerjee, Markus Graube, and Leon Urbas. 'NOA–Von Demonstratoren zu Pilotanwendungen: Vier Anwendungsfälle der Namur Open Architecture'. In: *atp magazin* 61.1-2 (2019), pp. 44–55.

[Dex] *DEXPI P&ID Specification Version 1.3*. Standard. DEXPI Initiative, June 2021. URL: `https://dexpi.org/wp-content/uploads/2020/09/DEXPI-PID-Specification-1.3.pdf`.

[Din] *DIN SPEC 91345:Referenzarchitekturmodell Industrie4.0 (RAMI4.0)*. Standard. DIN Deutsches Institut für Normung e.V., 2016.

[DMT12] DMTF. 'Common Information Model (CIM) Infrastructure'. In: (Apr. 2012).

[Dra09] R. Drath. *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. VDI-Buch. Springer Berlin Heidelberg, 2009. ISBN: 9783642046742.

[DuC13] Bob DuCharme. *Learning SPARQL: querying and updating with SPARQL 1.1.* " O'Reilly Media, Inc.", 2013.

[Eme03] Emerson. *EMERSON PROCESS MANAGEMENT: Reducing operations and maintenance costs - White paper, September 2003.* 2003.

[Fay+17] Alexander Fay, Christian Diedrich, Martin Dubovy, Christian Eck, Constantin Hildebrandt, André Scholz, Tizian Schröder, and Ralf Wiegand. *Vorhandene Standards als semantische Basis für die Anwendung von Industrie 4.0 (SemAnz40).* ger. Tech. rep. Holstenhofweg 85, 22043 Hamburg: Universitätsbibliothek der Helmut-Schmidt-Universität, 2017.

[FMS04] Karl Friedrich Früh, Uwe Maier, and Dieter Schaudel. *Handbuch der Prozessautomatisierung.* Oldenbourg Verlag-Strohmann, G.: Automatisierungstechnik (2 Bände . . ., 2004.

[Fou17] OPC Foundation. *OPC Unified Architecture - Interoperability for Industrie 4.0 and the Internet of Things.* OPC Foundation, 2017, p. 16. URL: `https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf`.

[Fou21a] OPC Foundation. *OPC 30081: OPC UA for Process Automation Devices – PA-DIM.* July 2021.

[Fou21b] OPC Foundation. *OPC 30090: OPC UA for Field Device Tool (FDT).* Aug. 2021.

[Fou21c] OPC Foundation. *OPC 30250: OPC Unified Architecture for DEXPI.* Sept. 2021.

[Fou22] OPC Foundation. *OPC 30080: OPC UA for Field Device Integration (FDI).* Apr. 2022.

[Fra+18] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 'Cypher: An evolving query language for property graphs'. In: *Proceedings of the 2018 International Conference on Management of Data.* 2018, pp. 1433–1445.

[GH20] Iris Graessler and Julian Hentze. 'The new V-Model of VDI 2206 and its validation'. In: *at-Automatisierungstechnik* 68.5 (2020), pp. 312–324.

[Gom11] H. Gomaa. *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures.* Cambridge University Press, 2011. ISBN: 9781139494731.

[GR89] Adele Goldberg and D. Robson. *Smalltalk-80: The Language.* Addison-Wesley series in computer science. Addison-Wesley, 1989. ISBN: 9780201136883.

[Gro+13] D. Großmann, M. Braun, B. Danzer, and M. Riedl. *FDI - Field Device Integration: Handbuch für die einheitliche Integrationstechnologie.* Vde Verlag GmbH, 2013. ISBN: 9783800735136.

[HAC14] Ernie Hayden, Michael Assante, and Tim Conway. 'An Abbreviated History of Automation and Industrial Controls Systems and Cybersecurity'. In: (Aug. 2014).

[Hen00] Ken Henderson. *The guru's guide to Transact-SQL.* Addison-Wesley Professional, 2000.

[HML81] Michael Hammer and Dennis Mc Leod. 'Database description with SDM: A semantic database model'. In: *ACM Transactions on Database Systems (TODS)* 6.3 (1981), pp. 351–386.

[HP18] Olaf Hartig and Jorge Pérez. 'Semantics and complexity of GraphQL'. In: *Proceedings of the 2018 World Wide Web Conference.* 2018, pp. 1155–1164.

[ISO09] ISO. 'ISO/TS 29002-5: 2009 Industrial automation systems and integration—Exchange of characteristic data—Part 5: Identification scheme'. In: (2009).

[Jar+92] Matthias Jarke, John Mylopoulos, Joachim W. Schmidt, and Yannis Vassiliou. 'DAIDA: An environment for evolving information systems'. In: *ACM Transactions on Information Systems (TOIS)* 10.1 (1992), pp. 1–50.

[Jes+16] S. Jeschke, C. Brecher, H. Song, and D.B. Rawat. *Industrial Internet of Things: Cybermanufacturing Systems.* Springer Series in Wireless Technology. Springer International Publishing, 2016. ISBN: 9783319425597.

[JNW10] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk.* Elsevier Science, 2010. ISBN: 9780080553849.

[Kag+13] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry ; Final Report of the Industrie 4.0 Working Group.* Forschungsunion, 2013.

[Keg07] G Kegel. 'Automation und Lifecycle Management Zusammenhänge und Abgrenzungen'. In: *Gesellschaft Mess-und Automatisierungstechnik (GMA)-Kongress, Baden-Baden.* 2007.

[Kha09] IIT Kharagpur. *Introduction to Industrial Automation and Control.* 2009. ISBN: 9780080553849. URL: `https://nptel.ac.in/courses/108105063/pdf/L-01(SM)(IA&C)%20((EE)NPTEL).pdf`.

[Kle+17a] Christian Klettner, Thomas Tauchnitz, Ulrich Epple, Lars Nothdurft, Christian Diedrich, Tizian Schröder, Daniel Großmann, Suprateek Banerjee, Michael Krauß, Chris Iatrou, et al. 'Namur Open Architecture: Die Namur-Pyramide wird geöffnet für Industrie 4.0'. In: *atp magazin* 59.01-02 (2017), pp. 20–37.

[Mat+20] Selvine G Mathias, Sebastian Schmied, Daniel Grossmann, Ralph Klaus Müller, and Björn Mroß. 'A compliance testing structure for implementation of industry standards through opc ua'. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).* Vol. 1. IEEE. 2020, pp. 1091–1094.

[Mik14] Martin Mikusz. 'Towards an understanding of cyber-physical systems as industrial software-product-service systems'. In: *Procedia Cirp* 16 (2014), pp. 385–389.

[MLD09] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture.* Springer Science & Business Media, 2009.

[Mon14] László Monostori. 'Cyber-physical production systems: Roots, expectations and R&D challenges'. In: *Procedia Cirp* 17 (2014), pp. 9–13.

[Müh12] M. Mühlhause. *Konzept Zur Durchgängigen Nutzung Von Engineeringmodellen der Automation.* Logos Verlag Berlin, 2012. ISBN: 9783832596675.

[Myl98] John Mylopoulos. *Characterizing Information Modeling Techniques.* Ed. by Peter Bernus, Kai Mertins, and Günter Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 17–57. ISBN: 978-3-662-03526-9. DOI: `10.1007/978-3-662-03526-9_2`. URL: `https://doi.org/10.1007/978-3-662-03526-9_2`.

[NAM19] NAMUR. *NAMUR NA 35:2019-01-17 Engineering and execution of PCT projects in process industry.* Jan. 2019, p. 35.

[Ne16a] DIN Deutsches Institut für Normung e.V. *DIN SPEC 820-1: Normungsarbeit - Teil 1: Grundsätze.* Beuth Publishing, 2016.

[Ne16b] DIN Deutsches Institut für Normung e.V. *DIN SPEC 91345: Referenzarchitekturmodell Industrie 4.0 (RAMI4.0).* 2016.

[OF19a] VDMA OPC Foundation. *OPC 40010-1: OPC UA for Robotics - Vertical Integration.* July 2019.

[OF19b] VDMA OPC Foundation. *OPC 40100-1: OPC UA for Machine Vision - Control, configuration management, recipe management, result management.* Aug. 2019.

[OF21] VDMA OPC Foundation. *OPC 40223: OPC UA for Pumps and Vacuum Pumps).* May 2021.

[OMG06] OMG. 'Meta Object Facility (MOF) Core Specification'. In: (2006).

[OT17] M. Oppitz and P. Tomsu. *Inventing the Cloud Century: How Cloudiness Keeps Changing Our Life, Economy and Technology.* Springer International Publishing, 2017. ISBN: 9783319611617.

[PH04] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, Third Edition.* The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2004. ISBN: 9780080502571.

[PU10] PLCOpen and OPC UA. 'PLCOpen and OPC Foundation: OPC UA Information Model for IEC 61131-3'. In: (Feb. 2010).

[Rot16] Armin Roth. *Einführung und Umsetzung von Industrie 4.0: Grundlagen, Vorgehensmodell und Use Cases aus der Praxis.* Springer-Verlag, 2016.

[RQ+12] Chris Rupp, Stefan Queins, et al. *UML 2 glasklar: Praxiswissen für die UML-Modellierung.* Carl Hanser Verlag GmbH Co KG, 2012.

[Run19] Krishna Rungta. *UML 2.0: Learn UML in 1 Day.* Guru99, 2019.

[Sch07] B. Scholten. *The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing.* ISA, 2007. ISBN: 9780979234385.

[Sch+19] Rainer Schiekofer, Stephan Grimm, Maja Milicic Brandt, and Michael Weyrich. 'A formal mapping between OPC UA and the semantic web'. In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN).* Vol. 1. IEEE. 2019, pp. 33–40.

[Soo+18] Gabor Soos, Daniel Kozma, Ferenc Nandor Janky, and Pal Varga. 'IoT Device Lifecycle – A Generic Model and a Use Case for Cellular Mobile Networks'. In: Aug. 2018, pp. 176–183. DOI: 10.1109/FiCloud.2018.00033.

[SVC06] Thomas Stahl, Markus Völter, and Krzysztof Czarnecki. *Model-driven software development: technology, engineering, management.* John Wiley & Sons, Inc., 2006.

[TL08] Y Tina Lee. ‘Information Modeling: From Design to Implementation’. In: (Aug. 2008).

[Usl+12] M. Uslar, M. Specht, S. Rohjans, J. Trefke, and J.M. González. *The Common Information Model CIM: IEC 61968/61970 and 62325 - A practical introduction to the CIM*. Power Systems. Springer Berlin Heidelberg, 2012. ISBN: 9783642252150.

[VDI21] VDI/VDE. *VDI/VDE 2206: 2021-11 Development of mechatronic and cyber-physical systems*. VDI/VDE 2206. Beuth Publishing, 2021. URL: `https://www.beuth.de/en/technical-rule/vdi-vde-2206/342674320`.

[WT21] Jan Nicolas Weskamp and Juilee Tikekar. ‘An Industrie 4.0 compliant and self-managing OPC UA Aggregation Server’. In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2021, pp. 1–8.

[WW18] Lihui Wang and Xi Vincent Wang. *Cloud-based cyber-physical systems in manufacturing*. Springer, 2018.

[Zur16] R. Zurawski. *Integration Technologies for Industrial Automated Systems*. Industrial Information Technology. CRC Press, 2016. ISBN: 9781420009040.

List of Publications

[BG16] Suprateek Banerjee and Ing Daniel Großmann. 'An Electronic Device Description Language based approach for communication with dbms and file system in an industrial automation scenario'. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2016, pp. 1–4. DOI: `10.1109/ETFA.2016.7733682`.

[BG17] Suprateek Banerjee and Daniel Großmann. 'Aggregation of information models — An OPC UA based approach to a holistic model of models'. In: *2017 4th International Conference on Industrial Engineering and Applications (ICIEA)*. 2017, pp. 296–299. DOI: `10.1109/IEA.2017.7939225`.

[BG19] Suprateek Banerjee and Daniel Grossmann. 'OPC UA and Dynamic Web Services: A Generic Flexible Industrial Communication Approach'. In: ICCAE 2019. Perth, WN, Australia: Association for Computing Machinery, 2019, 114–117. ISBN: 9781450362870. DOI: `10.1145/3313991.3313996`. URL: `https://doi.org/10.1145/3313991.3313996`.

[Cai+19] Jan de Caigny, Thomas Tauchnitz, Ronny Becker, Christian Diedrich, Tizian Schröder, Daniel Großmann, Suprateek Banerjee, Markus Graube, and Leon Urbas. 'NOA–Von Demonstratoren zu Pilotanwendungen: Vier Anwendungsfälle der Namur Open Architecture'. In: *atp magazin* 61.1-2 (2019), pp. 44–55.

[Gro+14a] Daniel Grossmann, Suprateek Banerjee, Markus Bregulla, Dirk Schulz, and Roland Braun. 'Auf dem Weg zum Internet of Portals'. In: *atp magazin* 56.07-08 (2014), pp. 42–51.

[Gro+14b] D. Großmann, S. Banerjee, M. Bregulla, D. Schulz, and R. Braun. 'OPC UA server aggregation — The foundation for an internet of portals'. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. 2014, pp. 1–6. DOI: `10.1109/ETFA.2014.7005354`.

[Gro+16] D. Großmann, S. Banerjee, J. Kiesbauer, and S. Erben. *Predictive Maintenance auf der Basis von FDI und OPC UA*. AUTOMATION 2016: 17. Branchentreff der Mess- und Automatisierungstechnik. Düsseldorf: VDI Verlag, 2016, 77–78. ISBN: 978-3-18-102284-9. DOI: `10 . 51202 / 9783181022849 - 77`. URL: `https://doi.org/10.51202/9783181022849-77`.

[Kle+17b] Christian Klettner, Thomas Tauchnitz, Ulrich Epple, Lars Nothdurft, Christian Diedrich, Tizian Schröder, Daniel Großmann, Suprateek Banerjee, Michael Krauß, Chris Iatrou, et al. 'Namur Open Architecture: Die Namur-Pyramide wird geöffnet für Industrie 4.0'. In: *atp magazin* 59.01-02 (2017), pp. 20–37.

[Mir+17] Jorge Miranda, Suprateek Banerjee, Jorge Cabral, Daniel Grossmann, Christian F. Pedersen, and Stefan R. Wagner. 'Analysis of OPC unified architecture for healthcare applications'. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2017, pp. 1–4. DOI: `10.1109/ETFA.2017.8247771`.

[Not+18] Lars Nothdurft, Ulrich Epple, Tizian Schröder, Christian Diedrich, Daniel Grossmann, Suprateek Banerjee, Sebastian Schmied, Chris Paul Iatrou, Markus Graube, Leon Urbas, et al. 'NOA Demonstratoren Special'. In: *atp magazin* 60.01-02 (2018), pp. 44–69.

[Sch+20] Sebastian Schmied, Daniel Grossmann, Selvine G. Mathias, and Suprateek Banerjee. 'Vertical Integration via Dynamic Aggregation of Information in OPC UA'. In: *Intelligent Information and Database Systems*. Ed. by Pawel Sitek, Marcin Pietranik, Marek Krotkiewicz, and Chutimet Srinilta. Singapore: Springer Singapore, 2020, pp. 204–215. ISBN: 978-981-15-3380-8.

Curriculum Vitae

This page is only available in the print version due to the regulations of the Otto-von-Guericke University.

Declaration of Honor

„I hereby declare that I produced this thesis without prohibited external assistance and that none other than the listed references and tools have been used. I did not make use of any commercial consultant concerning graduation. A third party did not receive any nonmonetary perquisites neither directly nor indirectly for activities which are connected with the contents of the presented thesis.
All sources of information are clearly marked, including my own publications.
In particular I have not consciously:

- Fabricated data or rejected undesired results
- Misused statistical methods with the aim of drawing other conclusions than those warranted by the available data
- Plagiarized data or publications
- Presented the results of other researchers in a distorted way

I do know that violations of copyright may lead to injunction and damage claims of the author and also to prosecution by the law enforcement authorities. I hereby agree that the thesis may need to be reviewed with an electronic data processing for plagiarism.

This work has not yet been submitted as a doctoral thesis in the same or a similar form in Germany or in any other country. It has not yet been published as a whole."

Frankfurt am Main, 17.11.2023	Suprateek Banerjee
....	....
Place, Date	Suprateek Banerjee